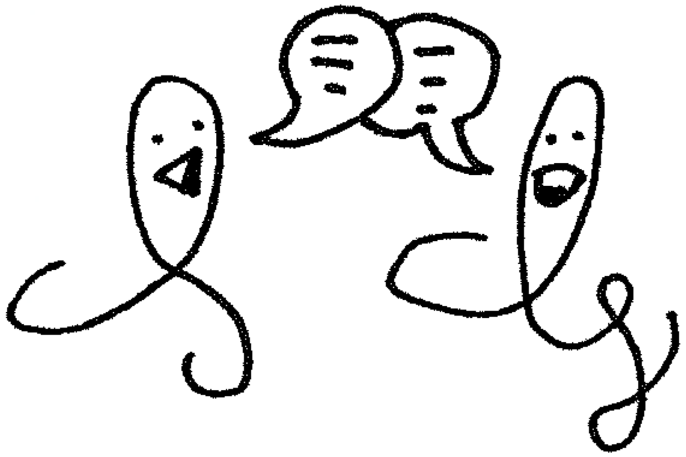
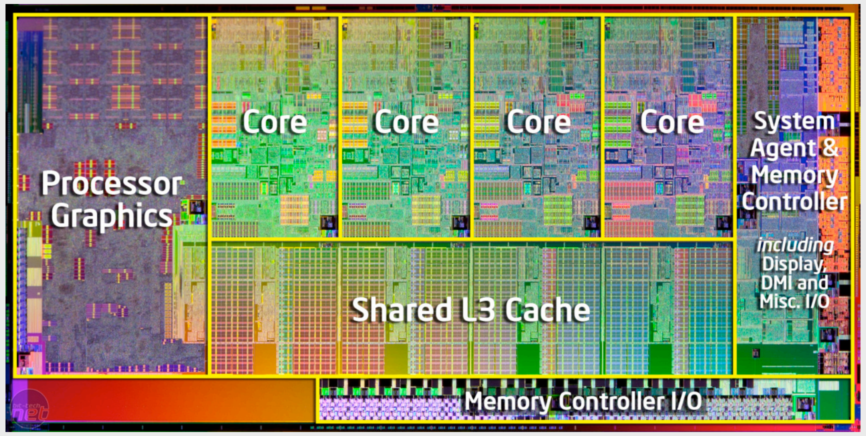
Abstractions for {**Expressive**, **Efficient**} Parallel and Distributed Computing

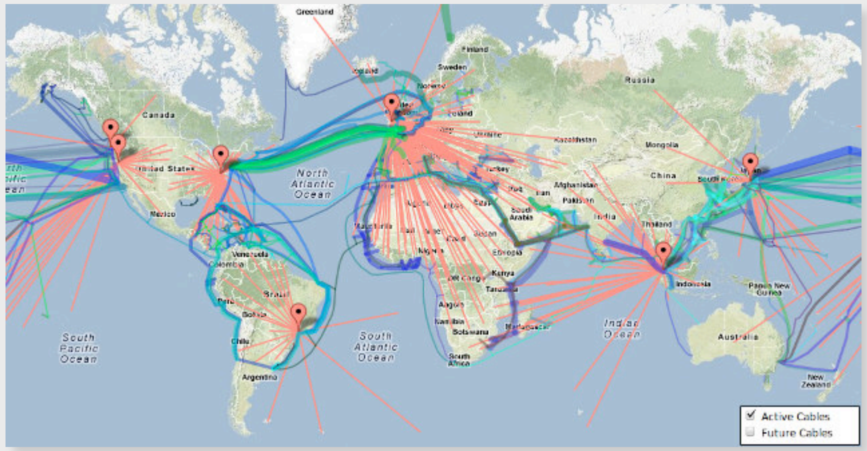


Lindsey Kuper
Assistant Professor, Computer Science and Engineering
UC Santa Cruz

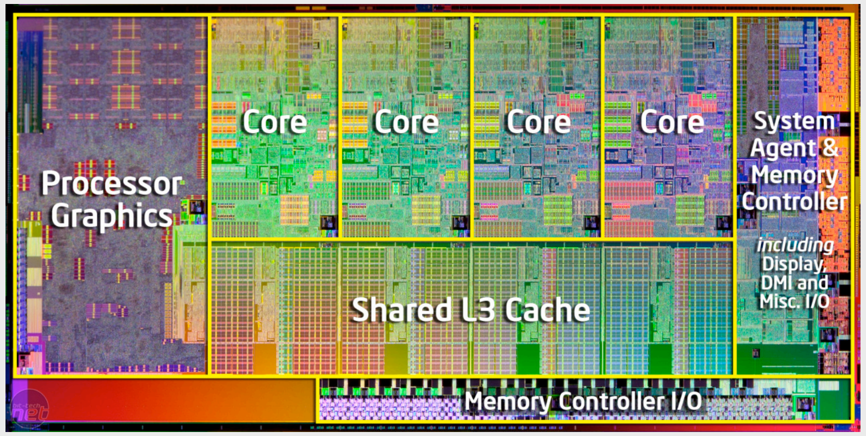
Jane Street Tech Talk
January 24, 2019

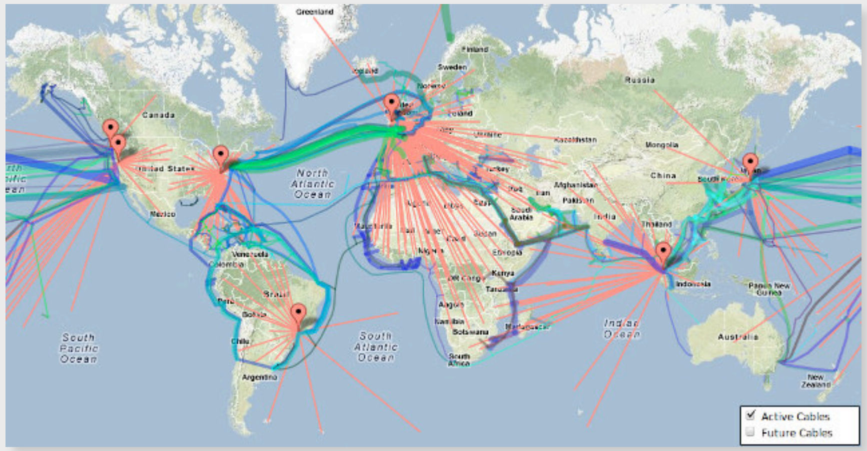


Parallel systems



Distributed systems

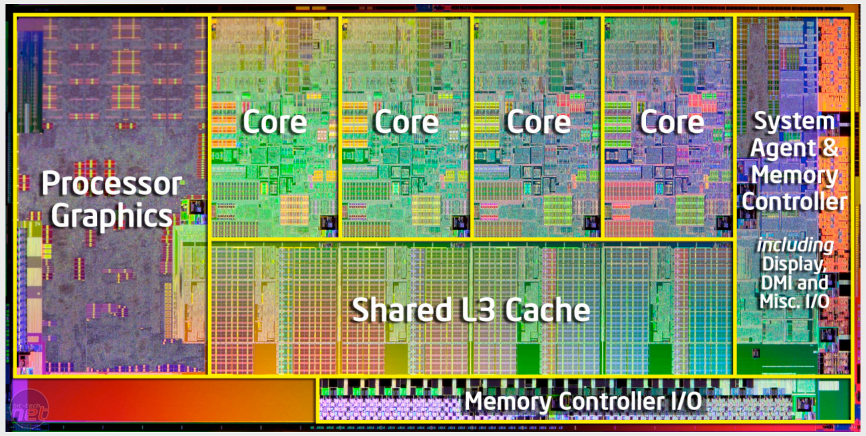


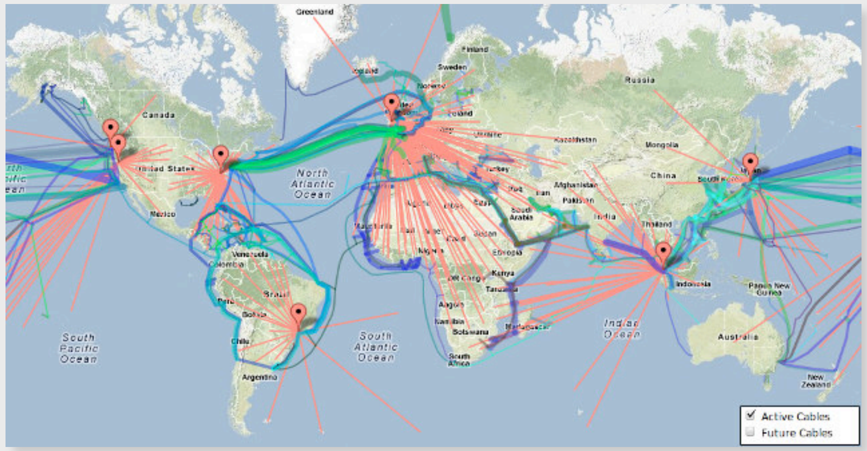


Parallel systems

Distributed systems

language-based approaches

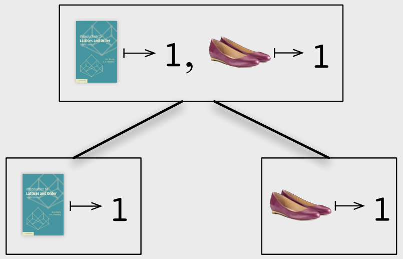


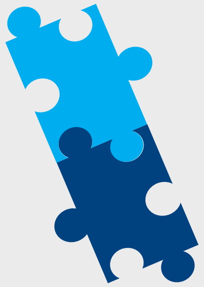


Parallel systems

Distributed systems

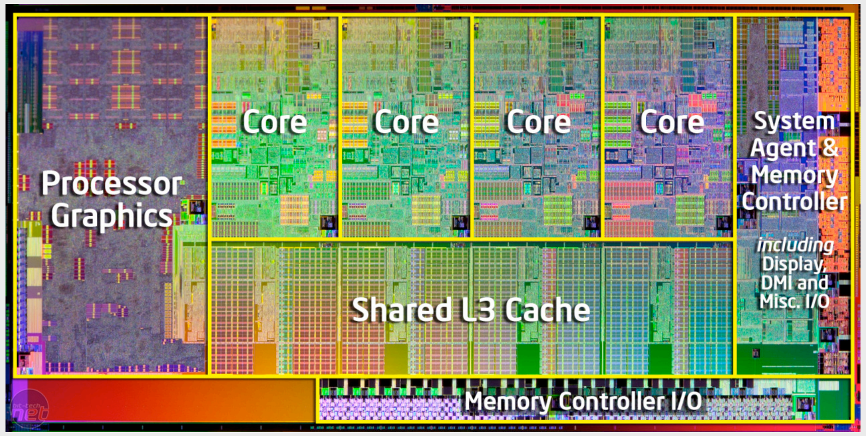
language-based approaches

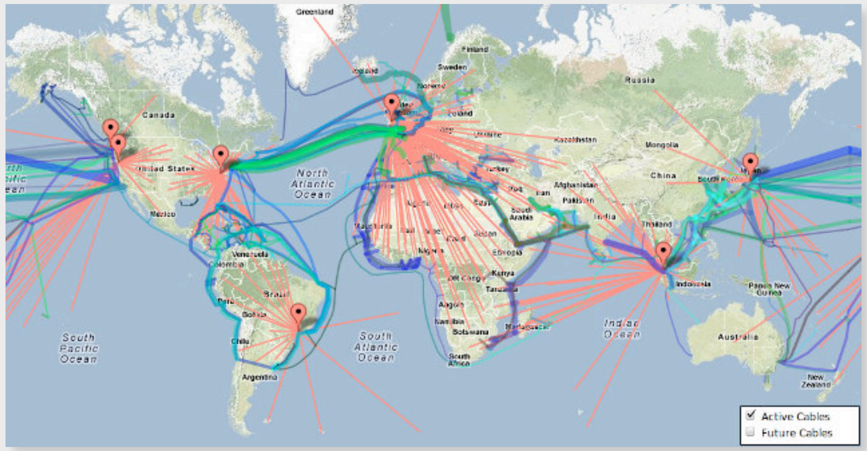




Lattice-based data structures (LVars) for deterministic programming
[POPL '14, PLDI '14, FHPC '13, WoDet '14]

Non-invasive DSLs for productive parallelism
[ECOOP '17]

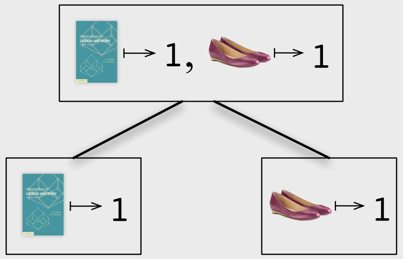


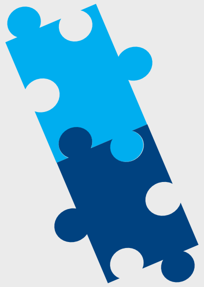


Parallel systems

Distributed systems

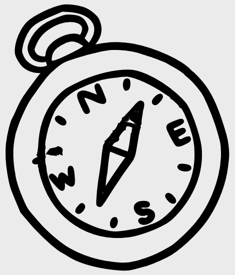
language-based approaches



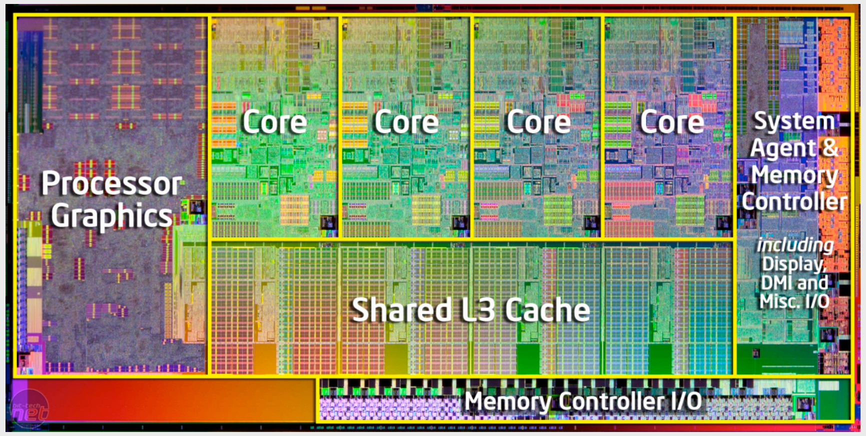


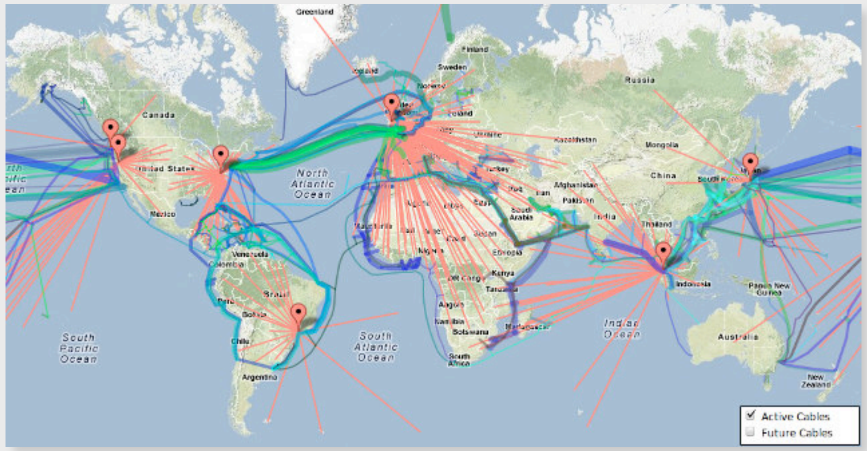
Lattice-based data structures (LVars) for deterministic programming
[POPL '14, PLDI '14, FHPC '13, WoDet '14]

Non-invasive DSLs for productive parallelism
[ECOOP '17]



Guiding principle:
Find the right **high-level abstractions** to enable **efficient** computation

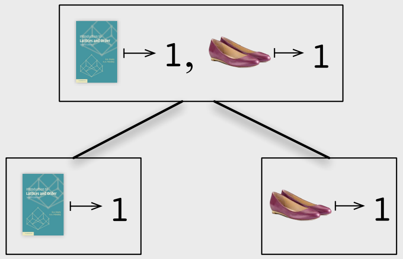




Parallel systems

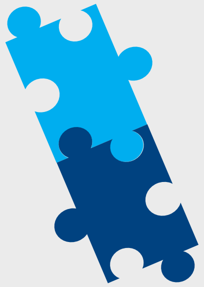
Distributed systems

language-based approaches



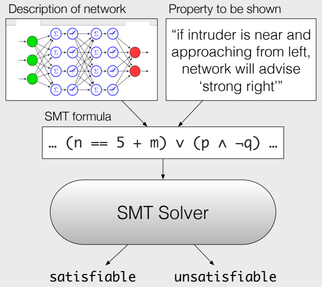
Lattice-based data structures (LVars) for deterministic programming

[POPL '14, PLDI '14, FHPC '13, WoDet '14]



Non-invasive DSLs for productive parallelism

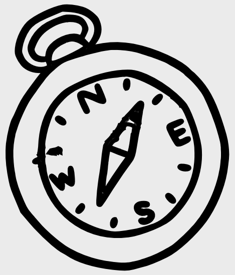
[ECOOP '17]



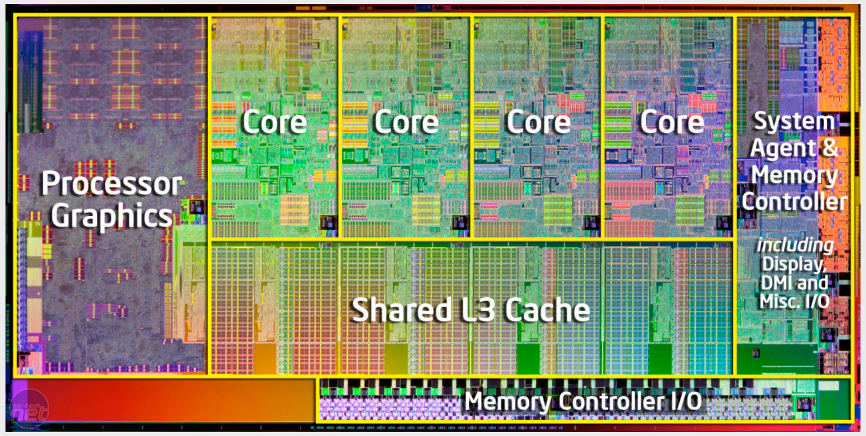
"if intruder is near and approaching from left, network will advise 'strong right'"

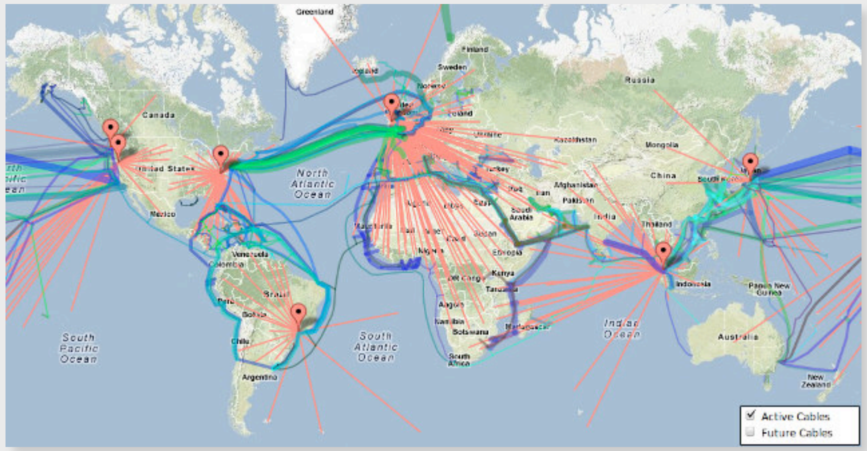
SMT-based verification of neural networks

[SysML '18]



Guiding principle:
Find the right **high-level abstractions**
to enable **efficient** computation

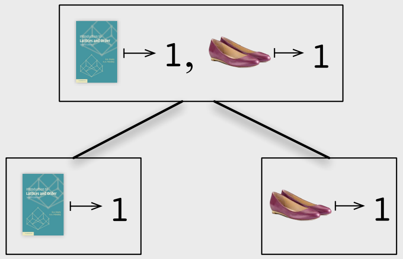


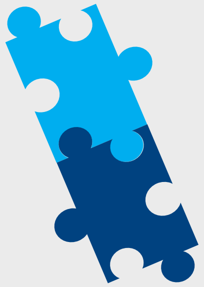


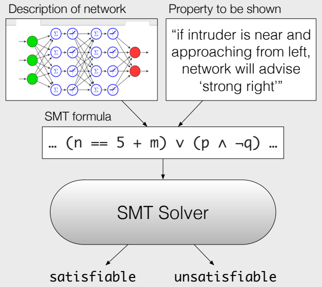
Parallel systems

Distributed systems

language-based approaches



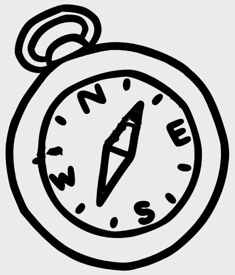



Description of network
Property to be shown
"if intruder is near and approaching from left, network will advise 'strong right'"
SMT formula
… (n == 5 + m) ∨ (p ∧ ¬q) …
SMT Solver
satisfiable
unsatisfiable

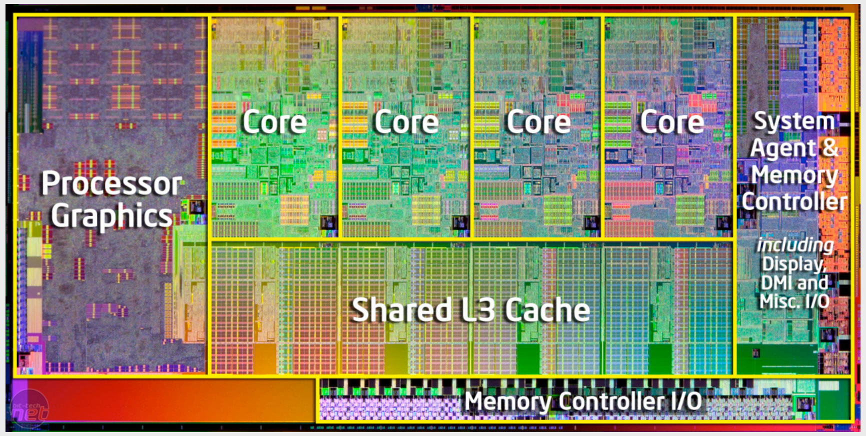
Lattice-based data structures (LVars) for deterministic programming
[POPL '14, PLDI '14, FHPC '13, WoDet '14]

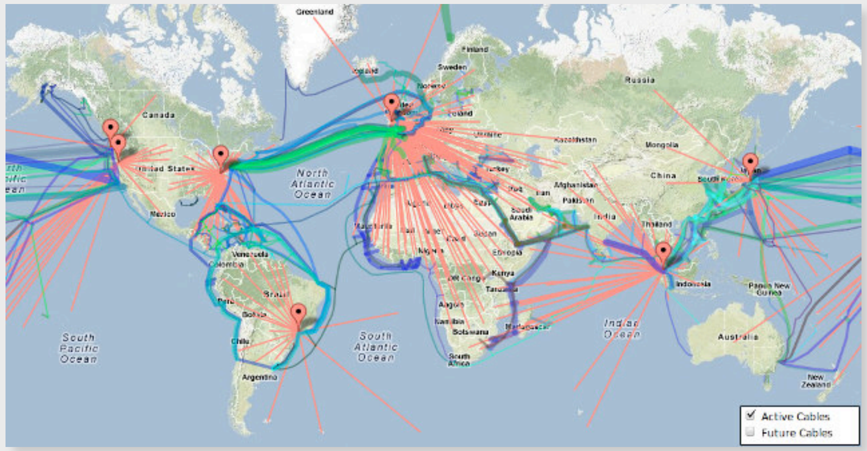
Non-invasive DSLs for productive parallelism
[ECOOP '17]

SMT-based verification of neural networks
[SysML '18]



Guiding principle:
Find the right **high-level abstractions**
to enable **efficient** computation

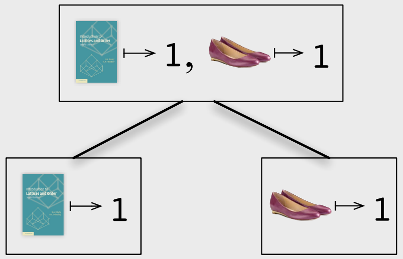




Parallel systems

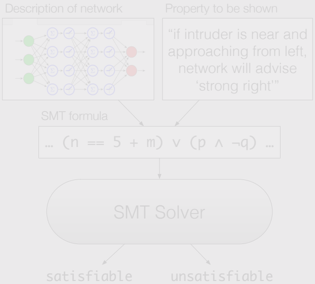
Distributed systems

language-based approaches



Lattice-based data structures (LVars) for deterministic programming
[POPL '14, PLDI '14, FHPC '13, WoDet '14]

Non-invasive DSLs for productive parallelism
[ECOOP '17]


Description of network
Property to be shown
"if intruder is near and approaching from left, network will advise 'strong right'"
SMT formula
… (n == 5 + m) ∨ (p ∧ ¬q) …
SMT Solver
satisfiable
unsatisfiable

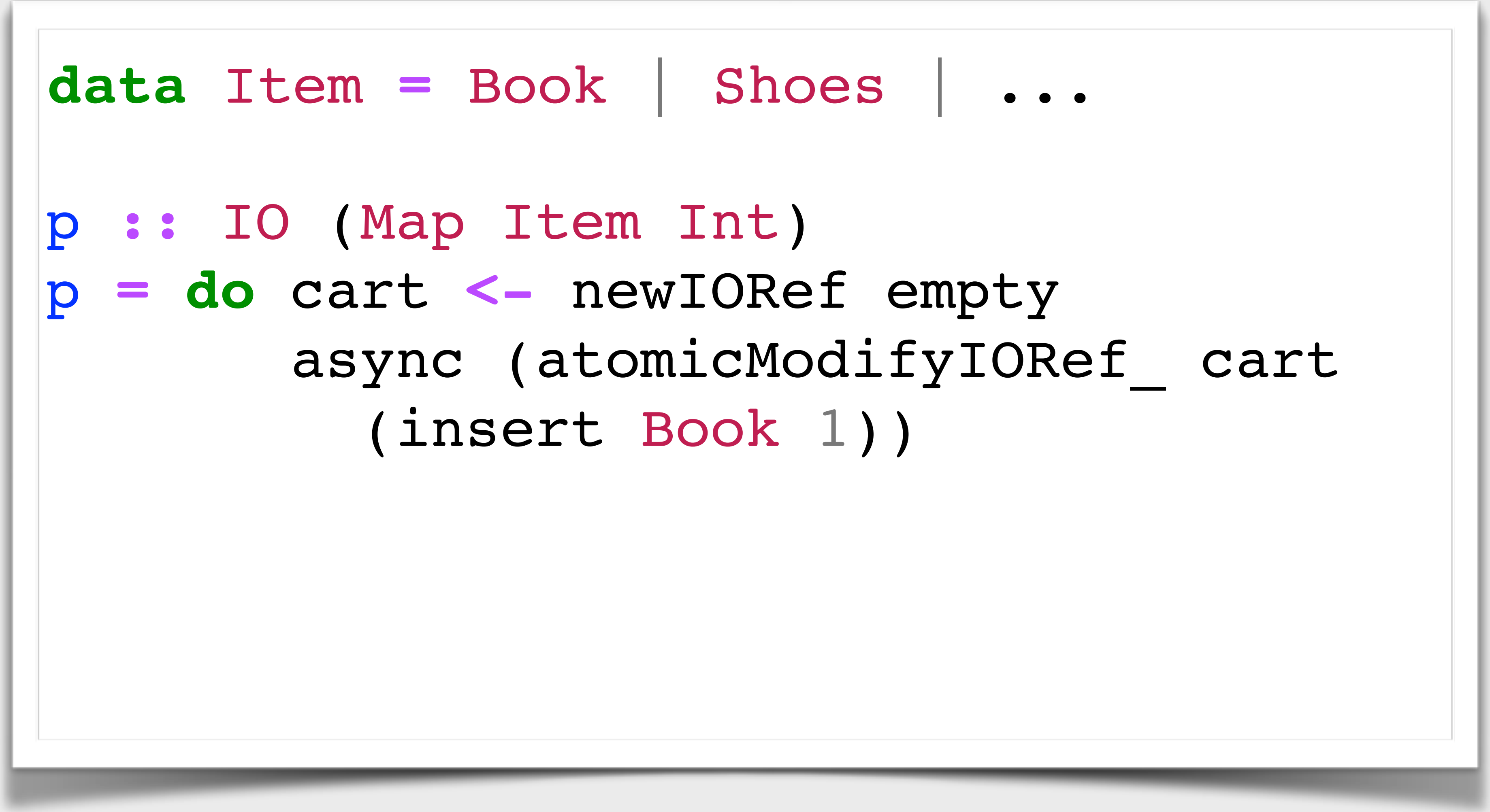
SMT-based verification of neural networks
[SysML '18]

Guiding principle:
Find the right **high-level abstractions** to enable **efficient** computation

```
data Item = Book | Shoes | ...
```

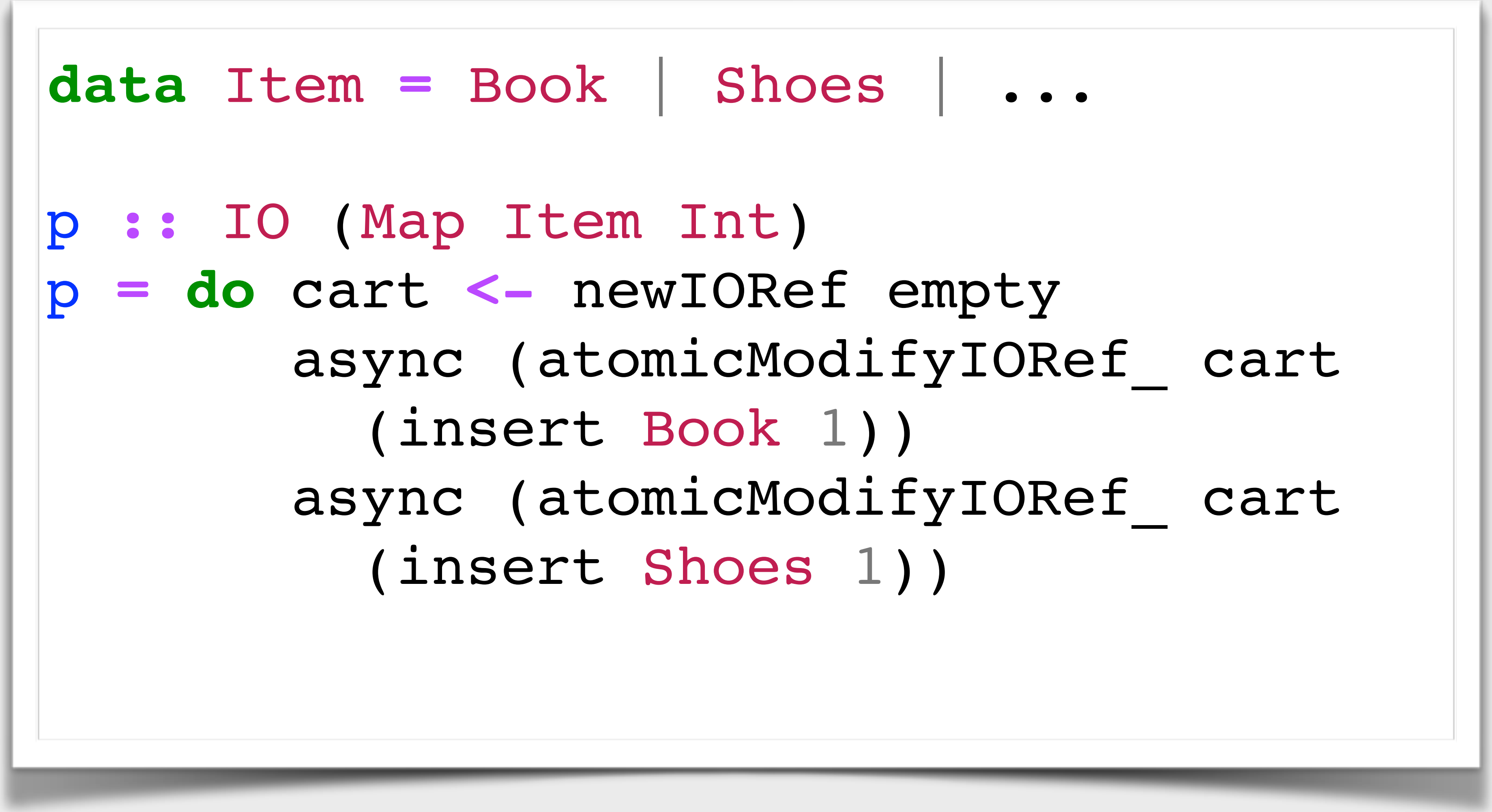
```haskell
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
```

```haskell
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
```

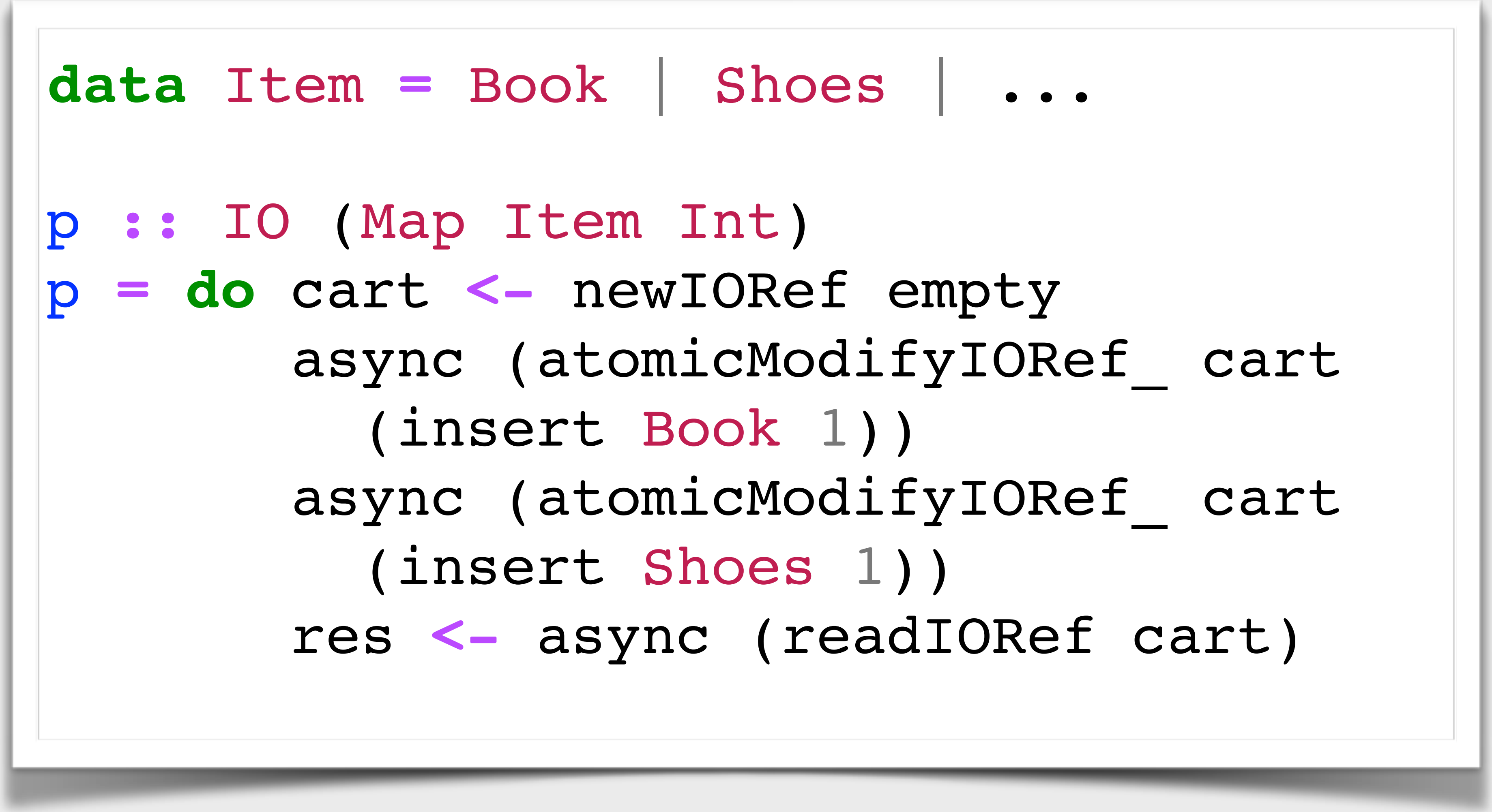


```
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
```

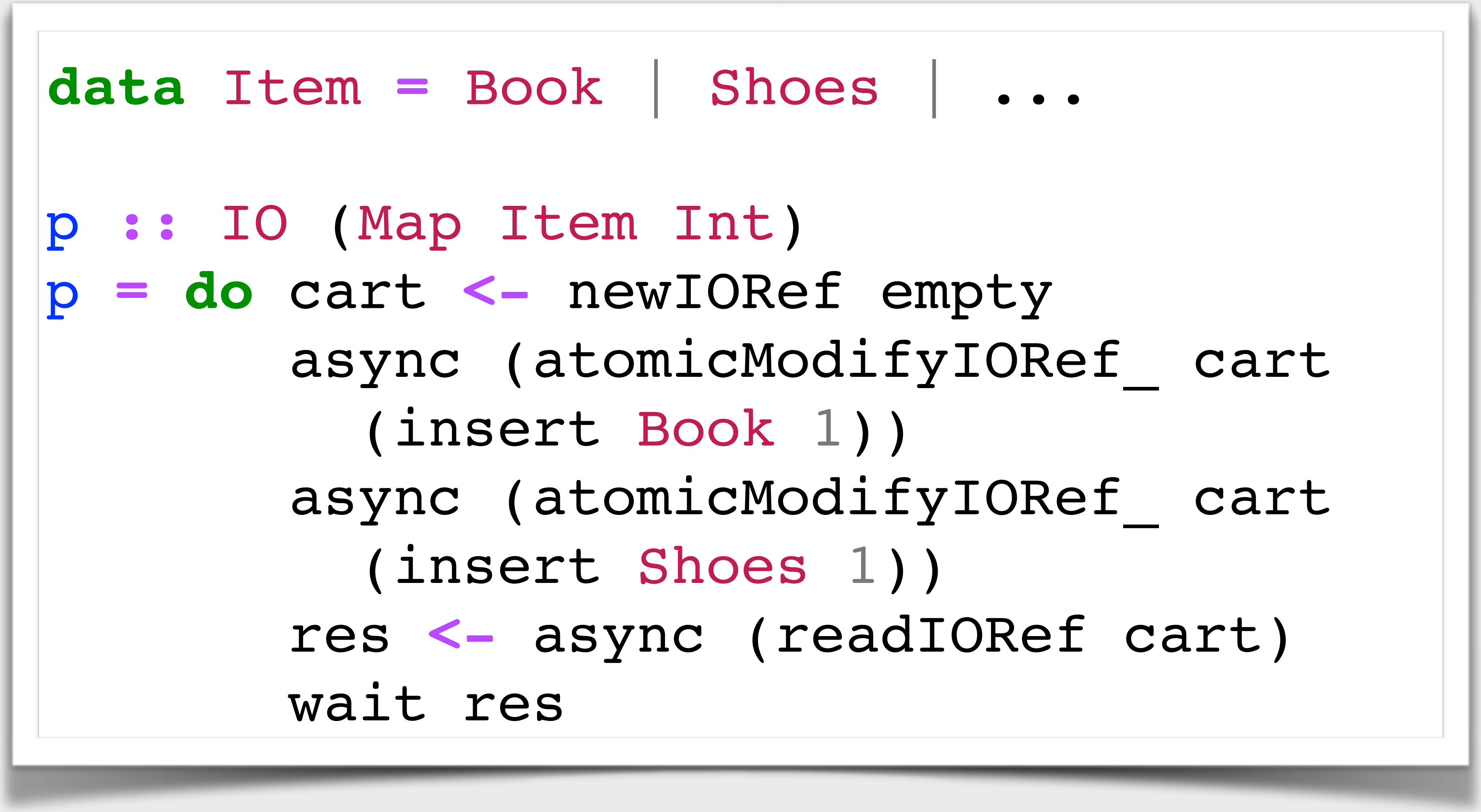
```
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
```



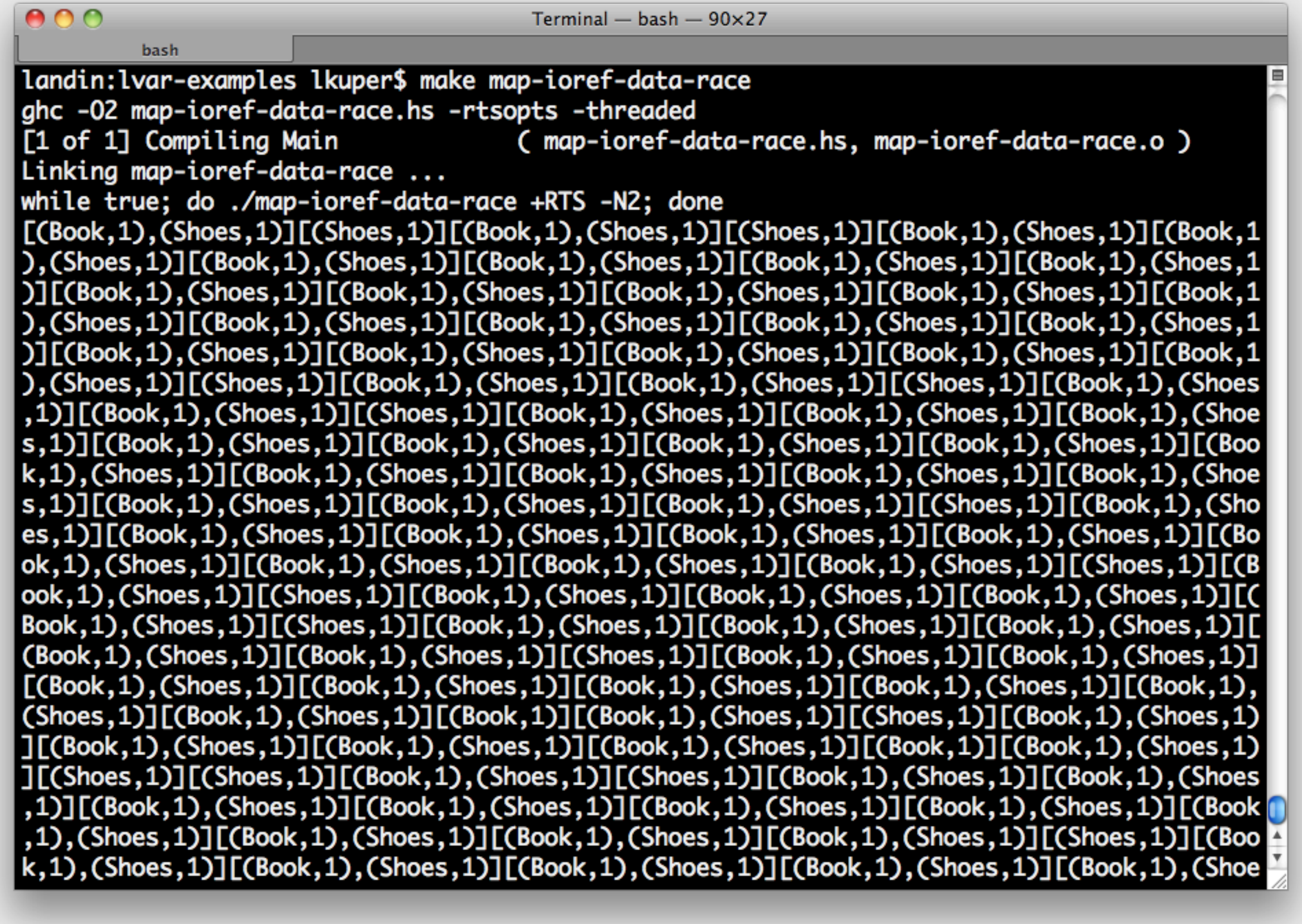
```haskell
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
       async (atomicModifyIORef_ cart
         (insert Shoes 1))
```



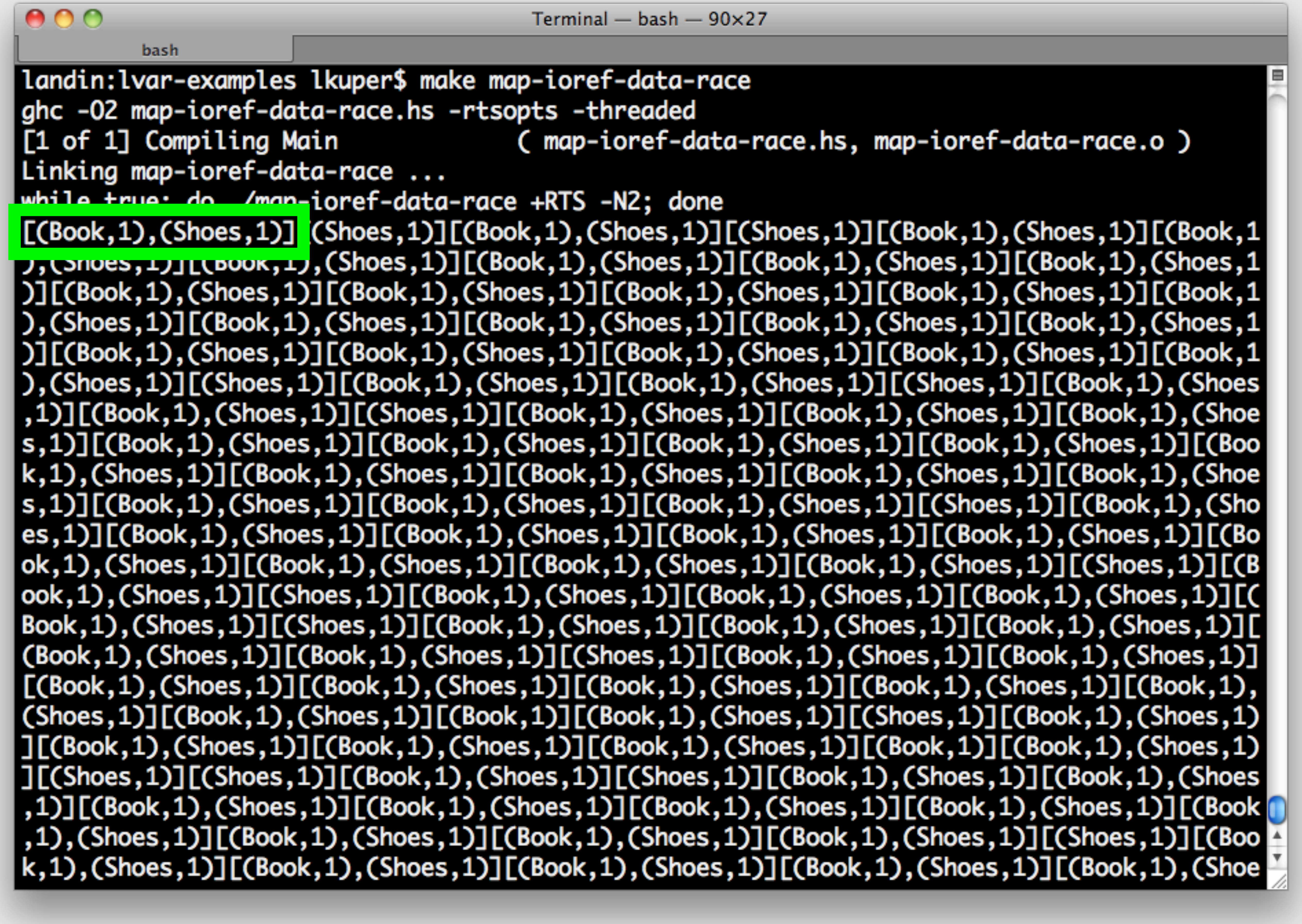
```haskell
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
       async (atomicModifyIORef_ cart
         (insert Shoes 1))
       res <- async (readIORef cart)
```



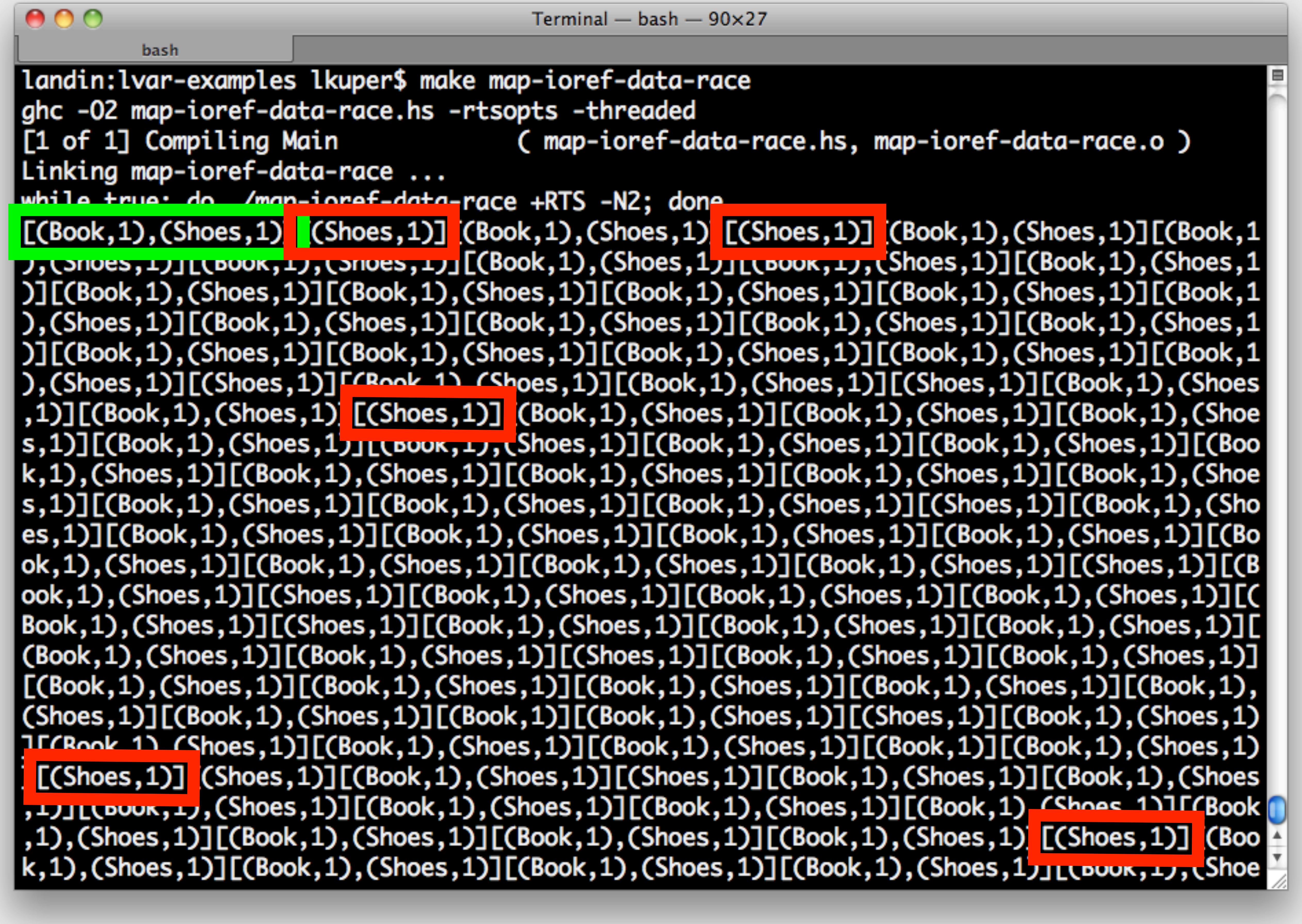
```
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
       async (atomicModifyIORef_ cart
         (insert Shoes 1))
       res <- async (readIORef cart)
       wait res
```



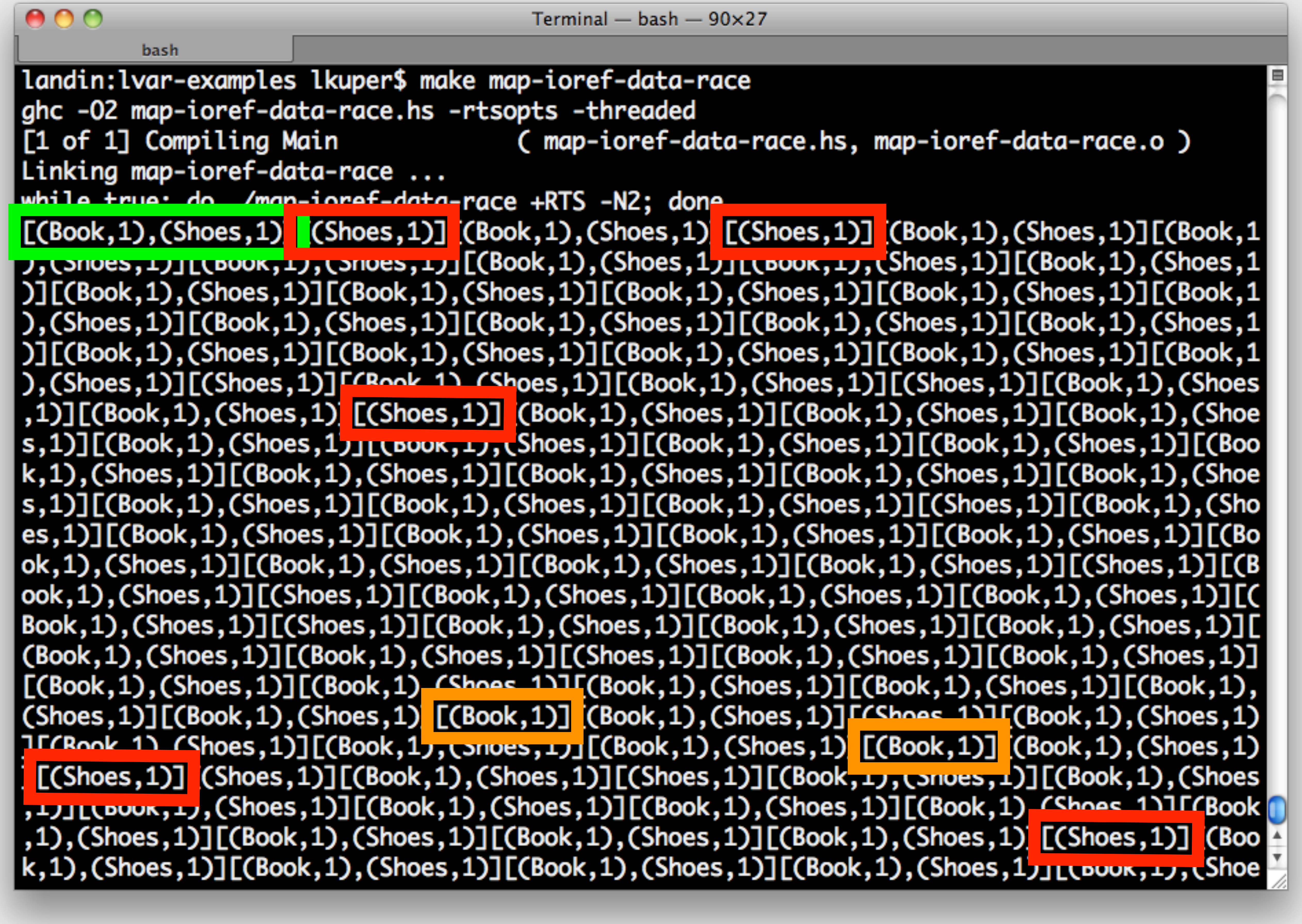
```
landin:lvar-examples lkuper$ make map-ioref-data-race
ghc -O2 map-ioref-data-race.hs -rtsopts -threaded
[1 of 1] Compiling Main             ( map-ioref-data-race.hs, map-ioref-data-race.o )
Linking map-ioref-data-race ...
while true; do ./map-ioref-data-race +RTS -N2; done
[(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1
),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1
)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1
),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1
)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1
),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes
,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoe
s,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Boo
k,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoe
s,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Sho
es,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Bo
ok,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(B
ook,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(
Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][
(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)]
[(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),
(Shoes,1)][(Book,1),(Shoes,1)][(Book,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)
][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1)][(Book,1),(Shoes,1)
][(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes
,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book
,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Boo
k,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoe
```



```
landin:lvar-examples lkuper$ make map-ioref-data-race
ghc -O2 map-ioref-data-race.hs -rtsopts -threaded
[1 of 1] Compiling Main             ( map-ioref-data-race.hs, map-ioref-data-race.o )
Linking map-ioref-data-race ...
while true; do ./map-ioref-data-race +RTS -N2; done
[(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1
),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1
)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1
),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1
)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1
),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes
,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoe
s,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Boo
k,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoe
s,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Sho
es,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Bo
ok,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(B
ook,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(
Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][
(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)]
[(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),
(Shoes,1)][(Book,1),(Shoes,1)][(Book,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)
][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1)][(Book,1),(Shoes,1)
][(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes
,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book
,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Boo
k,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoe
```



```
landin:lvar-examples lkuper$ make map-ioref-data-race
ghc -O2 map-ioref-data-race.hs -rtsopts -threaded
[1 of 1] Compiling Main             ( map-ioref-data-race.hs, map-ioref-data-race.o )
Linking map-ioref-data-race ...
while true; do ./map-ioref-data-race +RTS -N2; done
[(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1
),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1
)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1
),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1
)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1
),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes
,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoe
s,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Boo
k,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoe
s,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Sho
es,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Bo
ok,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(B
ook,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(
Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][
(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)]
[(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),
(Shoes,1)][(Book,1),(Shoes,1)][(Book,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)
][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1)][(Book,1),(Shoes,1)
][(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes
,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book
,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Boo
k,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoe
```



```
landin:lvar-examples lkuper$ make map-ioref-data-race
ghc -O2 map-ioref-data-race.hs -rtsopts -threaded
[1 of 1] Compiling Main             ( map-ioref-data-race.hs, map-ioref-data-race.o )
Linking map-ioref-data-race ...
while true; do ./map-ioref-data-race +RTS -N2; done
[(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1
),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1
)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1
),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1
)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1
),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes
,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoe
s,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Boo
k,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoe
s,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Sho
es,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Bo
ok,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(B
ook,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(
Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][
(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)]
[(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),
(Shoes,1)][(Book,1),(Shoes,1)][(Book,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)
][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1)][(Book,1),(Shoes,1)
][(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes
,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book
,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Shoes,1)][(Boo
k,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoes,1)][(Book,1),(Shoe
```

if we want determinism,
we have to *share nicely*

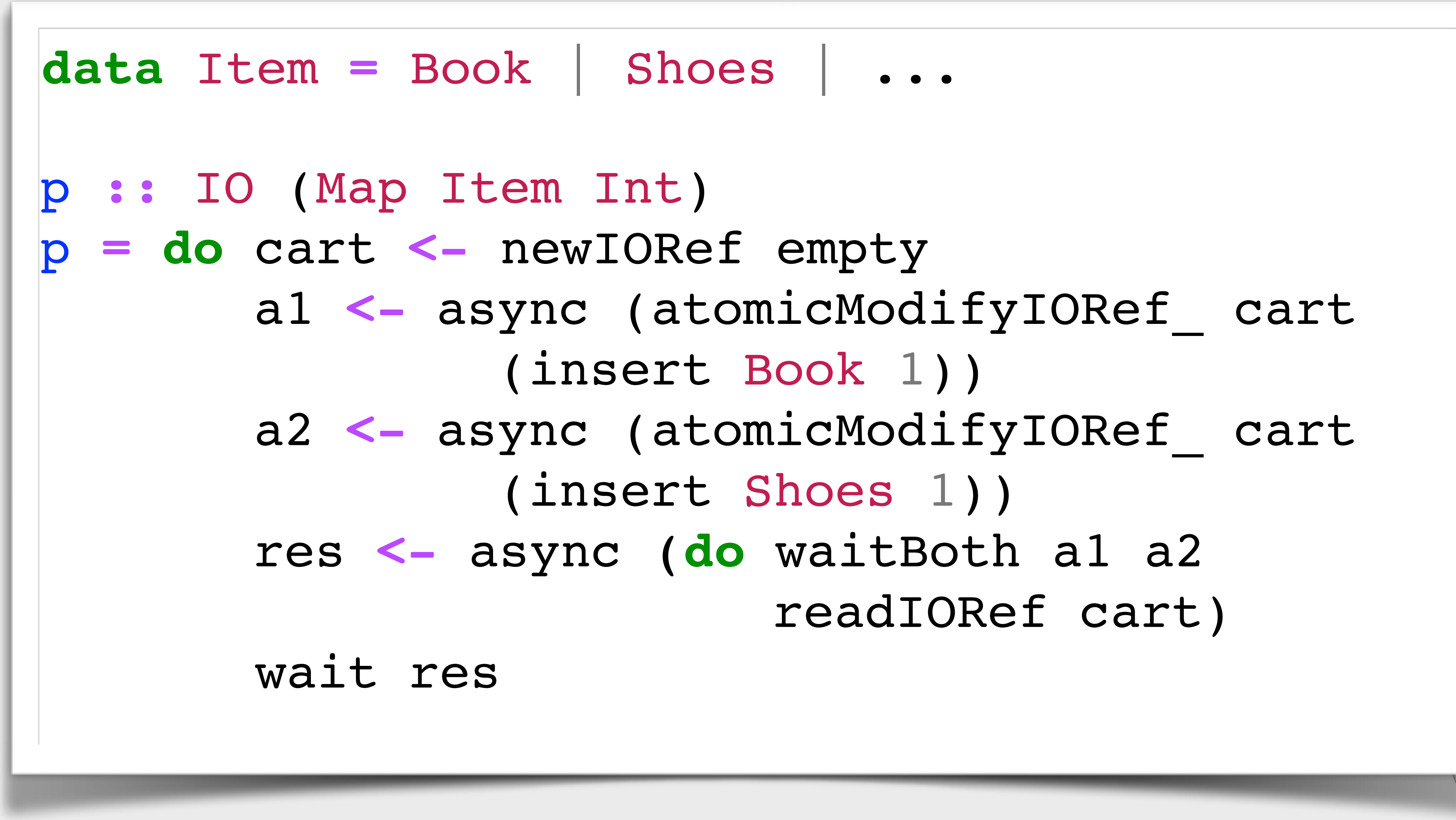
```haskell
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
       async (atomicModifyIORef_ cart
         (insert Shoes 1))
       res <- async (readIORef cart)
       wait res
```

```
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
       async (atomicModifyIORef_ cart
         (insert Shoes 1))
       res <- async (readIORef cart)
       wait res
```

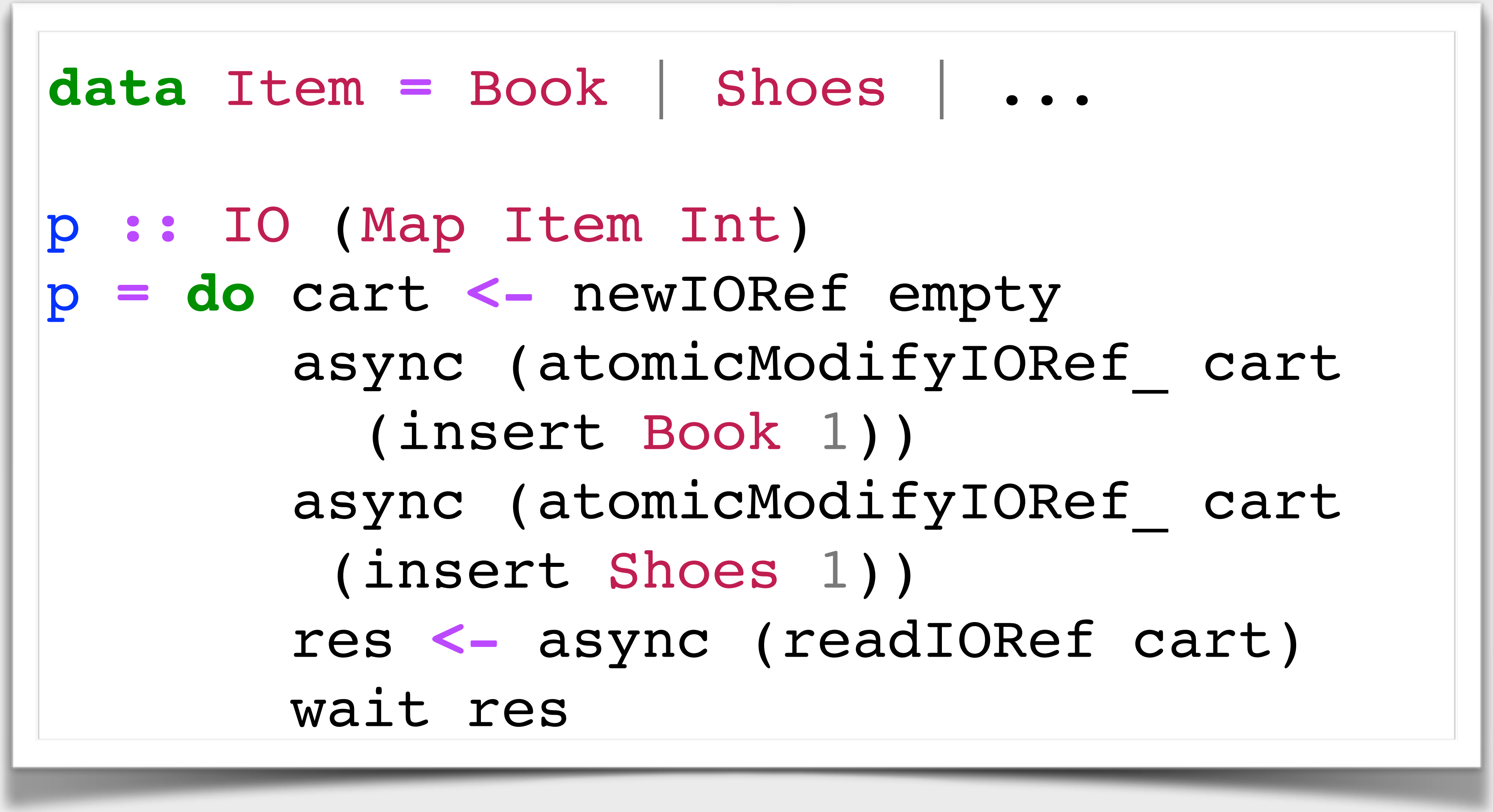
```haskell
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       a1 <- async (atomicModifyIORef_ cart
               (insert Book 1))
       async (atomicModifyIORef_ cart
         (insert Shoes 1))
       res <- async (readIORef cart)
       wait res
```

```haskell
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       a1 <- async (atomicModifyIORef_ cart
               (insert Book 1))
       a2 <- async (atomicModifyIORef_ cart
               (insert Shoes 1))
       res <- async (readIORef cart)
       wait res
```

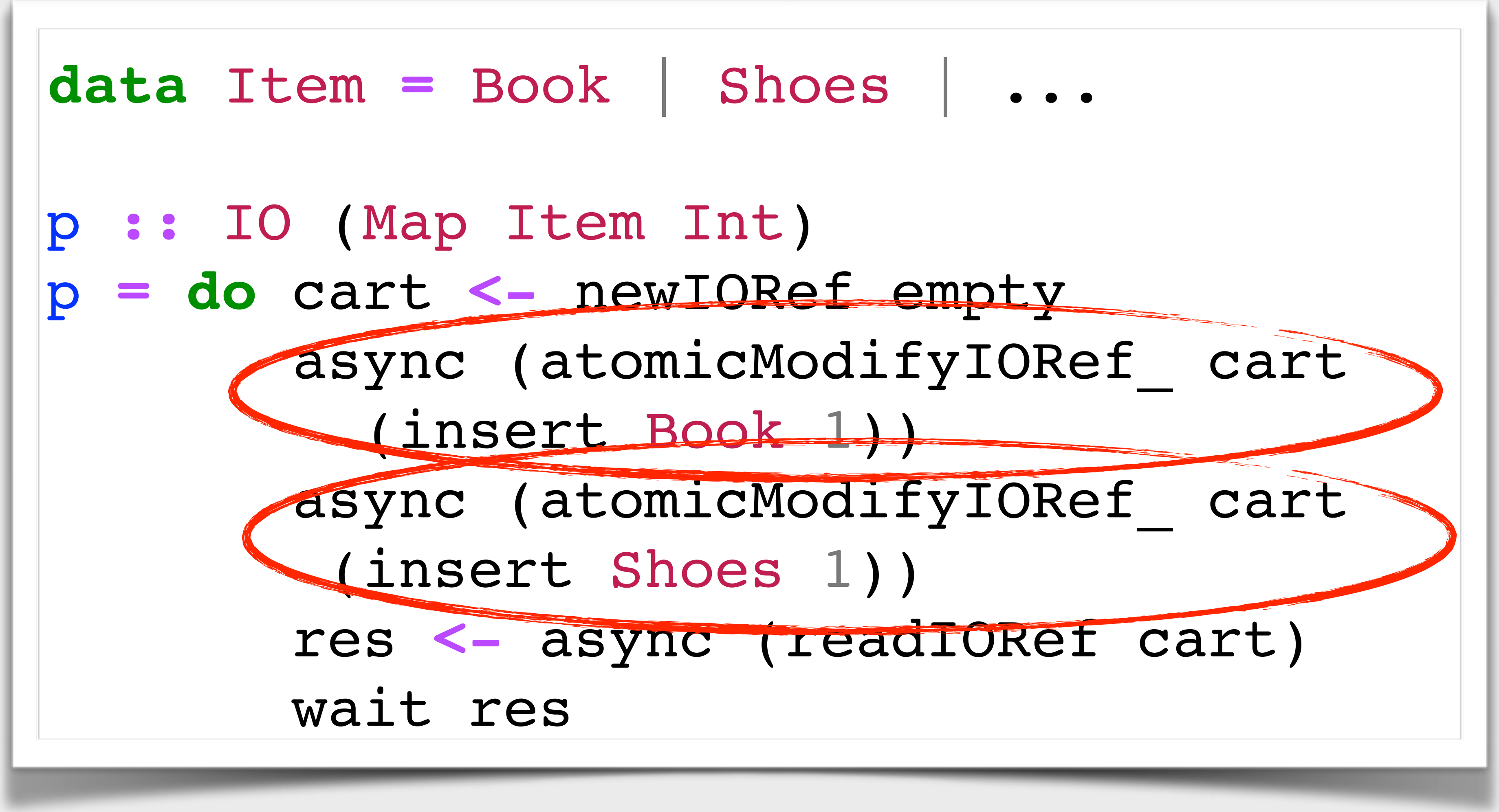
```
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       a1 <- async (atomicModifyIORef_ cart
               (insert Book 1))
       a2 <- async (atomicModifyIORef_ cart
               (insert Shoes 1))
       res <- async (do waitBoth a1 a2
       wait res         readIORef cart)
```



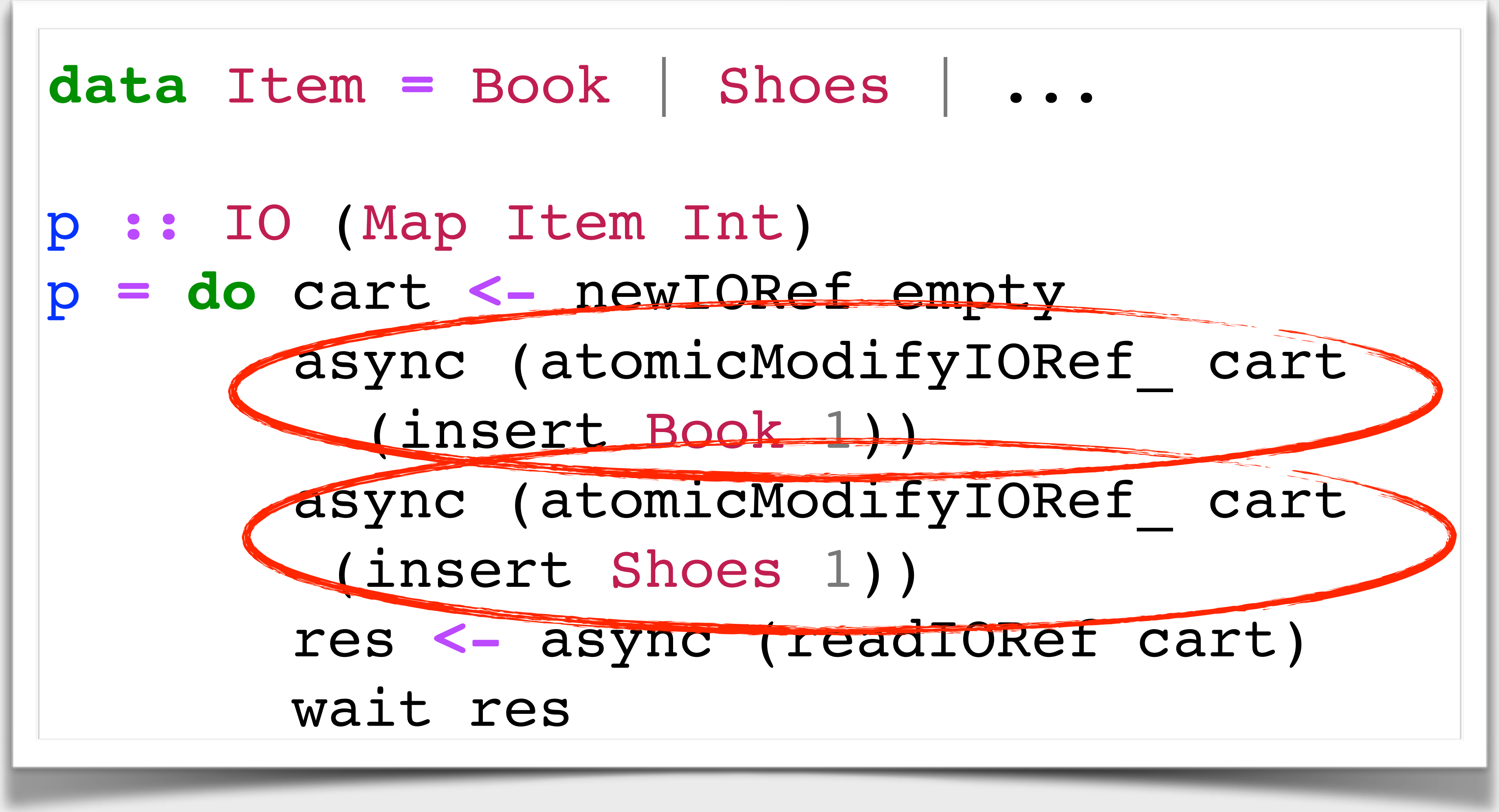
```
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       a1 <- async (atomicModifyIORef_ cart
               (insert Book 1))
       a2 <- async (atomicModifyIORef_ cart
               (insert Shoes 1))
       res <- async (do waitBoth a1 a2
                        readIORef cart)
       wait res
```



```haskell
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
       async (atomicModifyIORef_ cart
        (insert Shoes 1))
       res <- async (readIORef cart)
       wait res
```



```
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
       async (atomicModifyIORef_ cart
        (insert Shoes 1))
       res <- async (readIORef cart)
       wait res
```

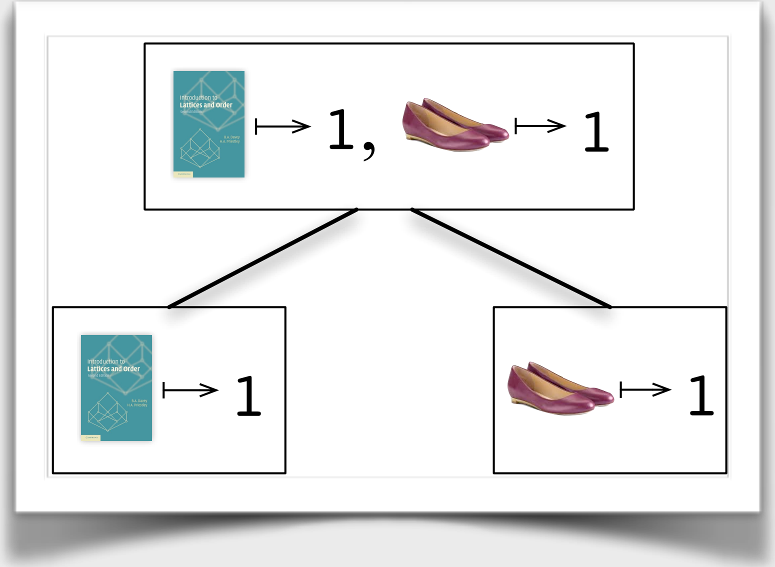


```
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
       async (atomicModifyIORef_ cart
        (insert Shoes 1))
       res <- async (readIORef cart)
       wait res
```

IVars: single writes, blocking (but exact) reads

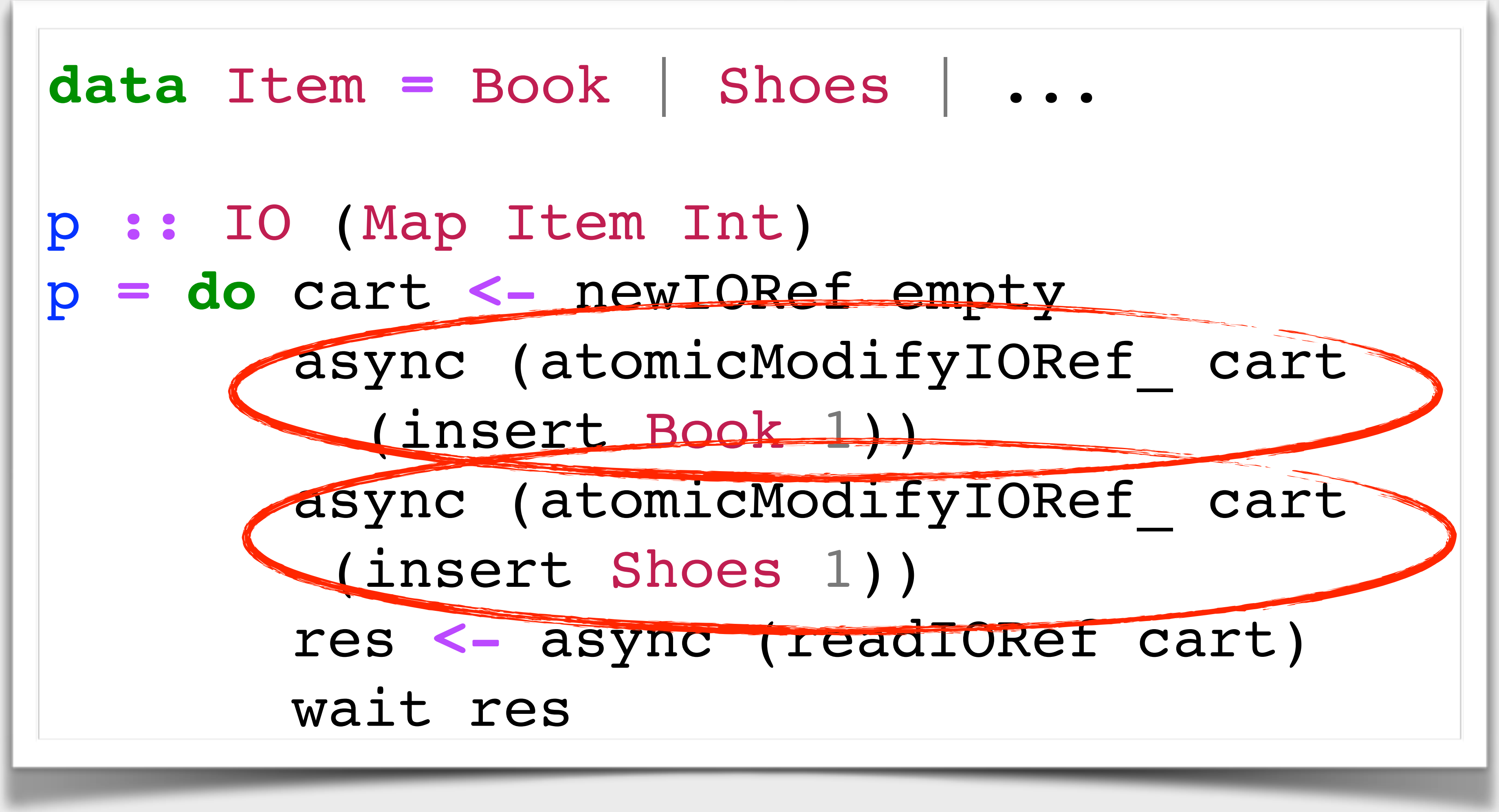
[Arvind *et al.*, 1989]

```haskell
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
       async (atomicModifyIORef_ cart
        (insert Shoes 1))
       res <- async (readIORef cart)
       wait res
```

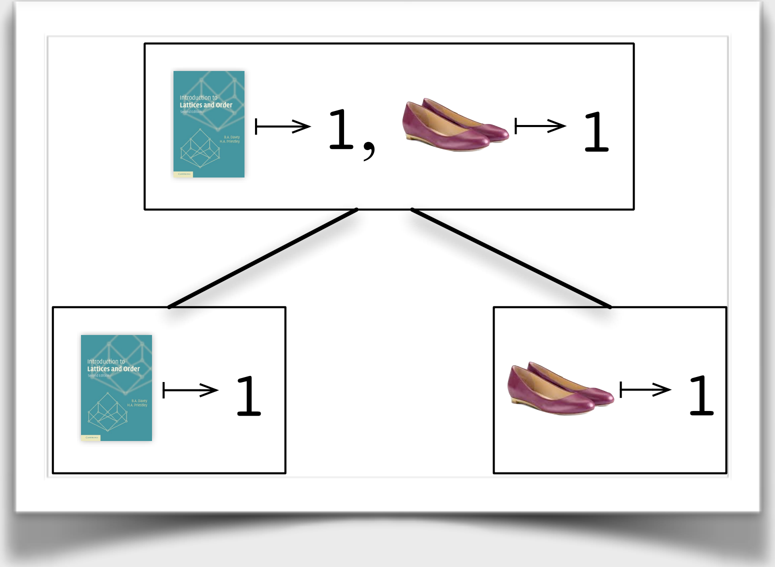


IVars: single writes, blocking (but exact) reads

[Arvind *et al.*, 1989]



```
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
       async (atomicModifyIORef_ cart
        (insert Shoes 1))
       res <- async (readIORef cart)
       wait res
```

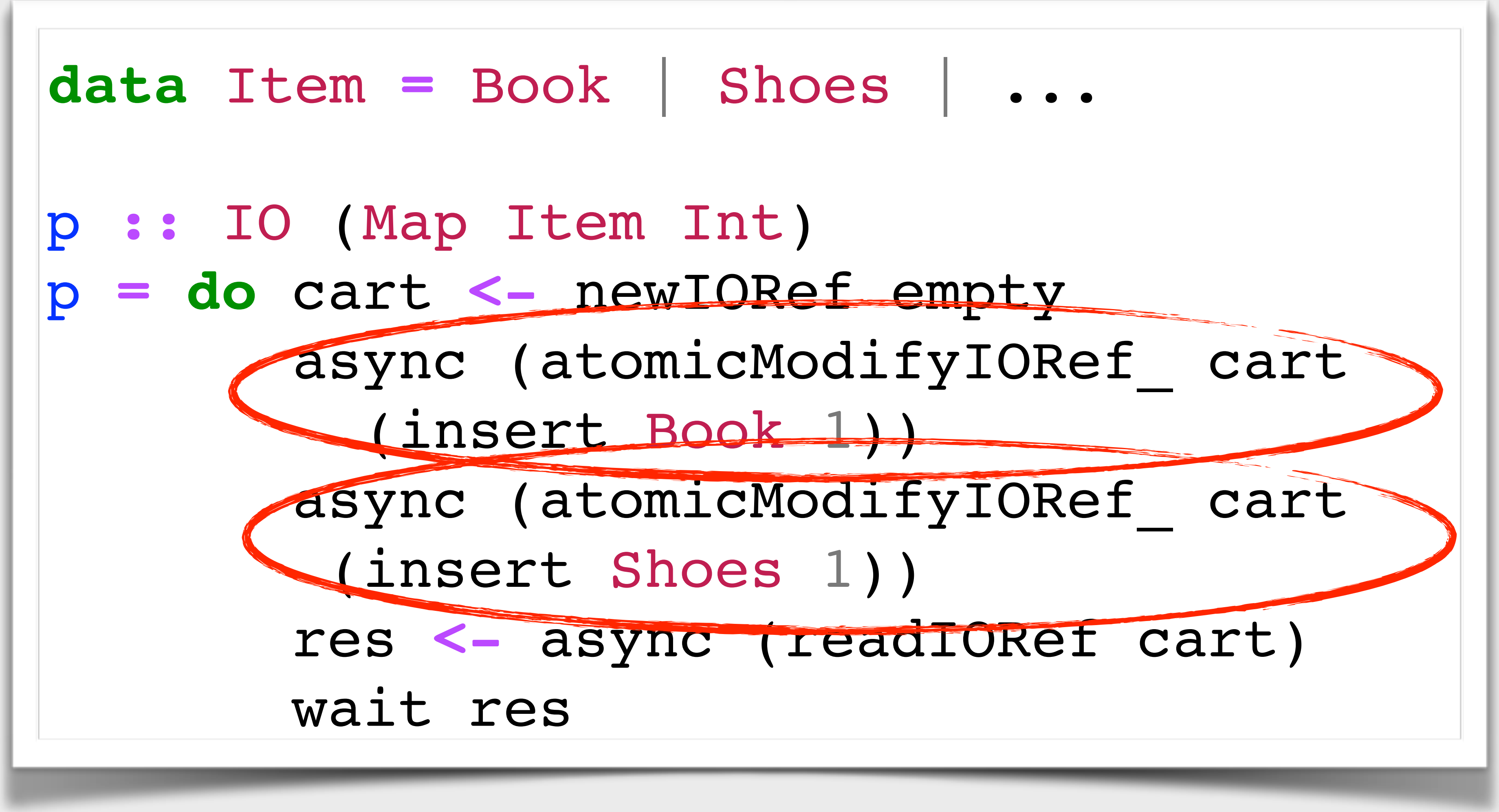


IVars: single writes, blocking (but exact) reads

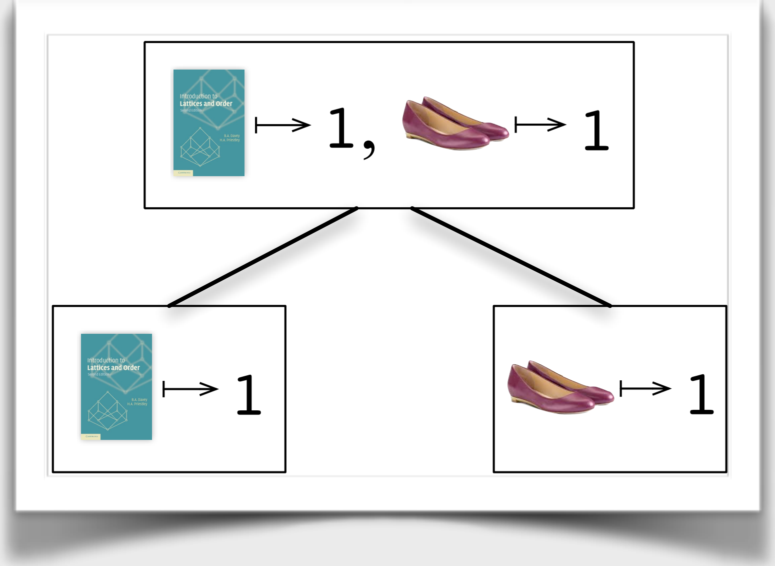
[Arvind *et al.*, 1989]

LVars: multiple *commutative* and *inflationary* writes, blocking *threshold* reads

[FHPC ’13]



```
data Item = Book | Shoes | ...

p :: IO (Map Item Int)
p = do cart <- newIORef empty
       async (atomicModifyIORef_ cart
         (insert Book 1))
       async (atomicModifyIORef_ cart
        (insert Shoes 1))
       res <- async (readIORef cart)
       wait res
```



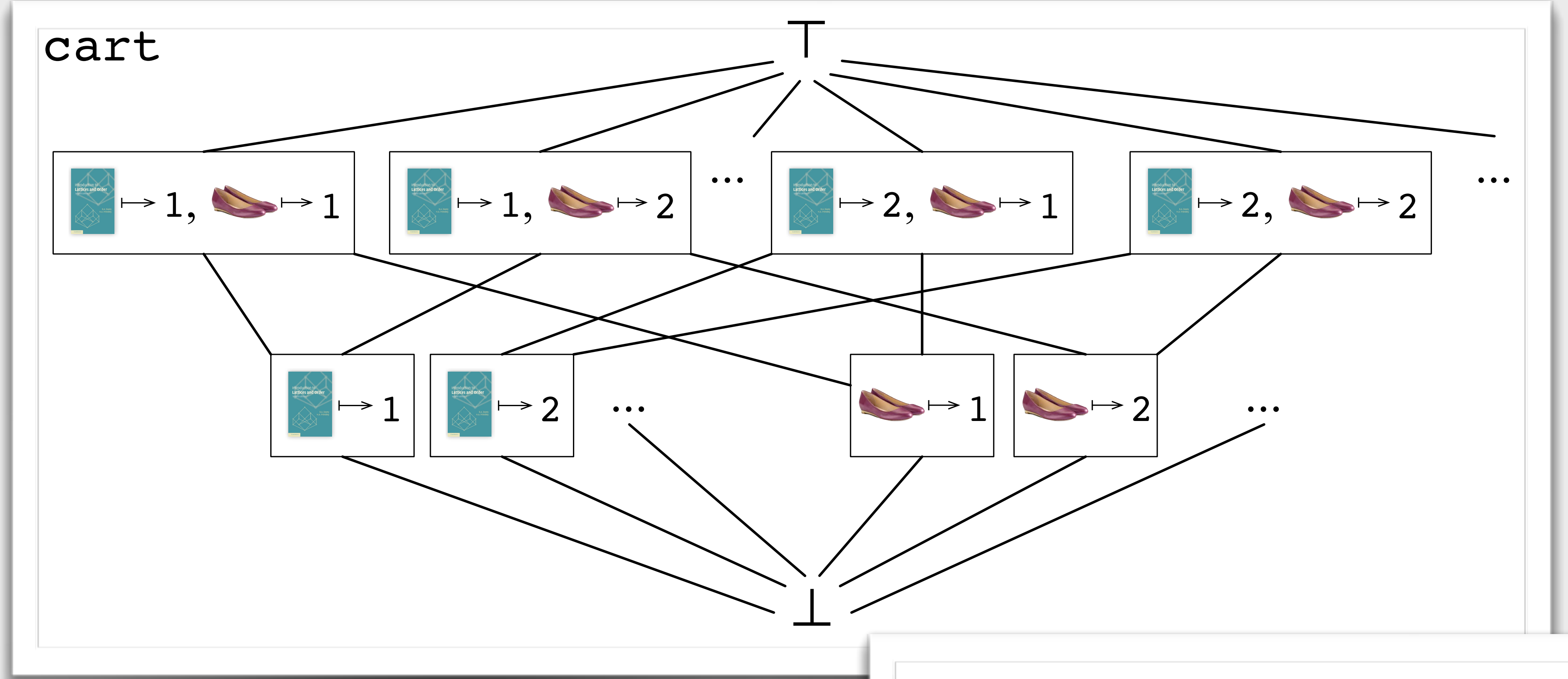
IVars: single writes, blocking (but exact) reads

[Arvind *et al.*, 1989]

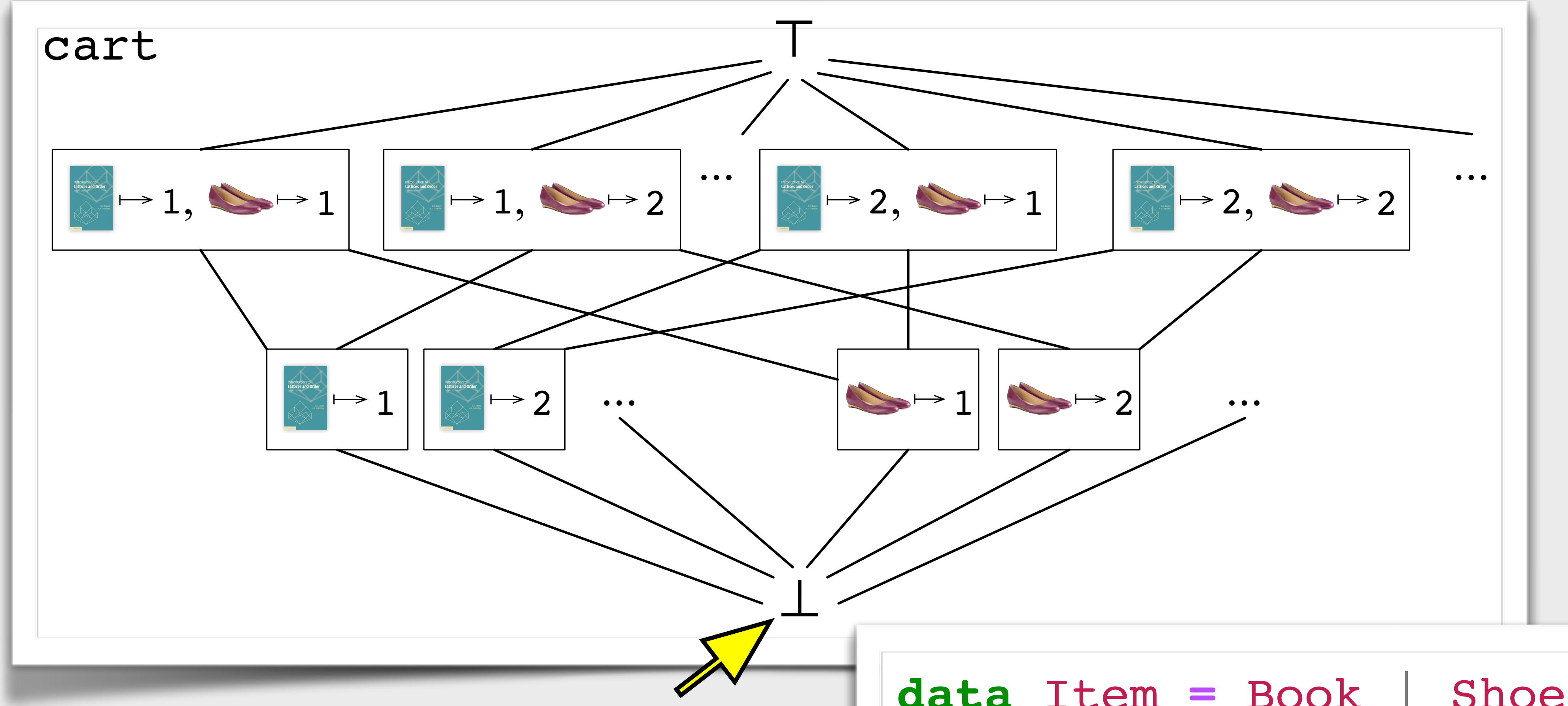
LVars: multiple *commutative* and *inflationary* writes, blocking *threshold* reads

[FHPC '13]

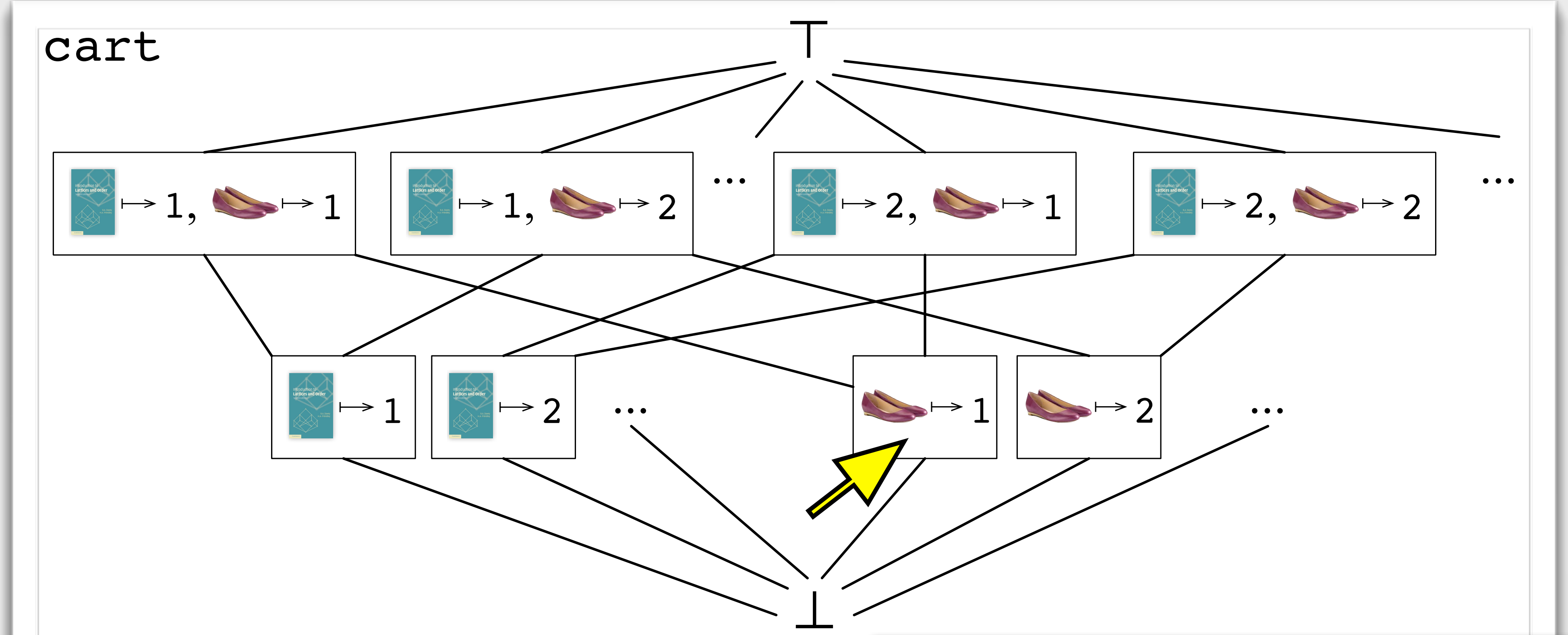
* actually a bounded join-semilattice



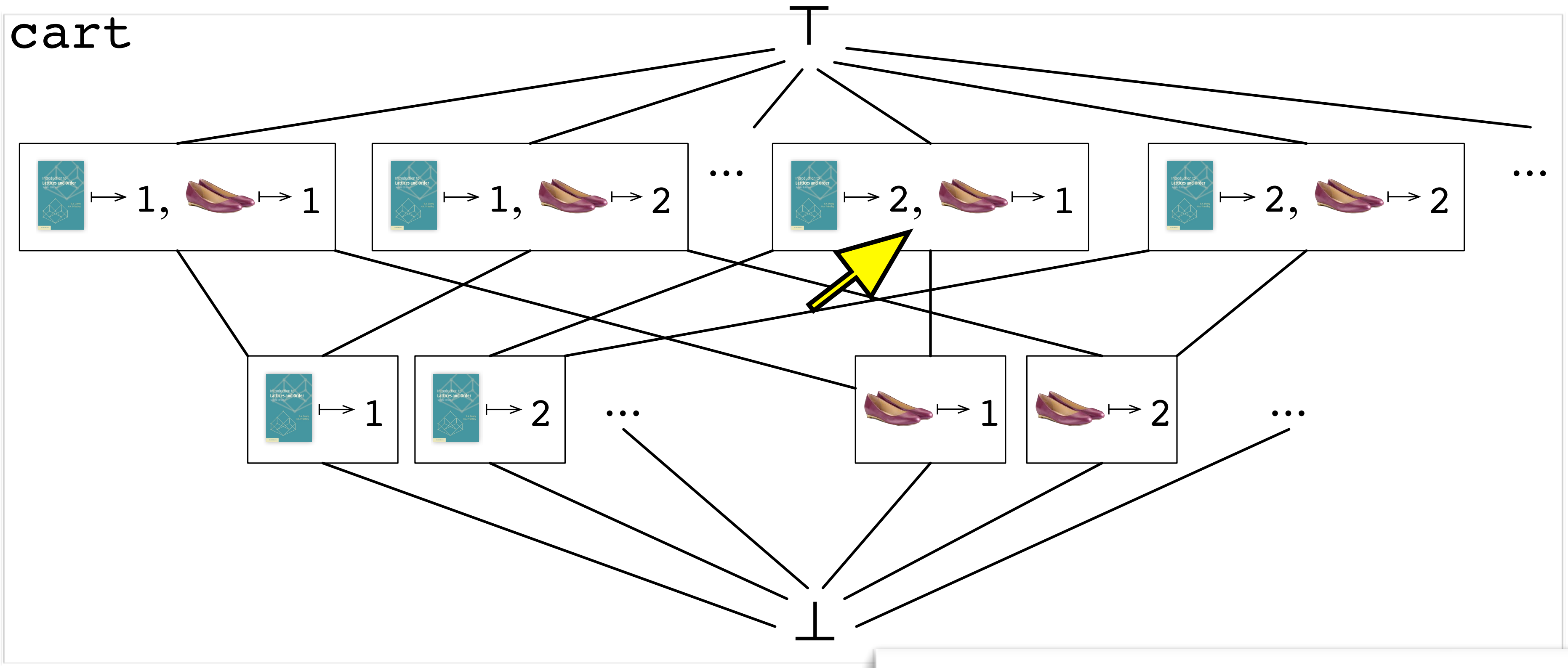
```
data Item = Book | Shoes | ...

p = do
  cart <- newEmptyMap
  fork (insert Shoes 1 cart)
  fork (insert Book 2 cart)
  getKey Book cart -- returns 2
```



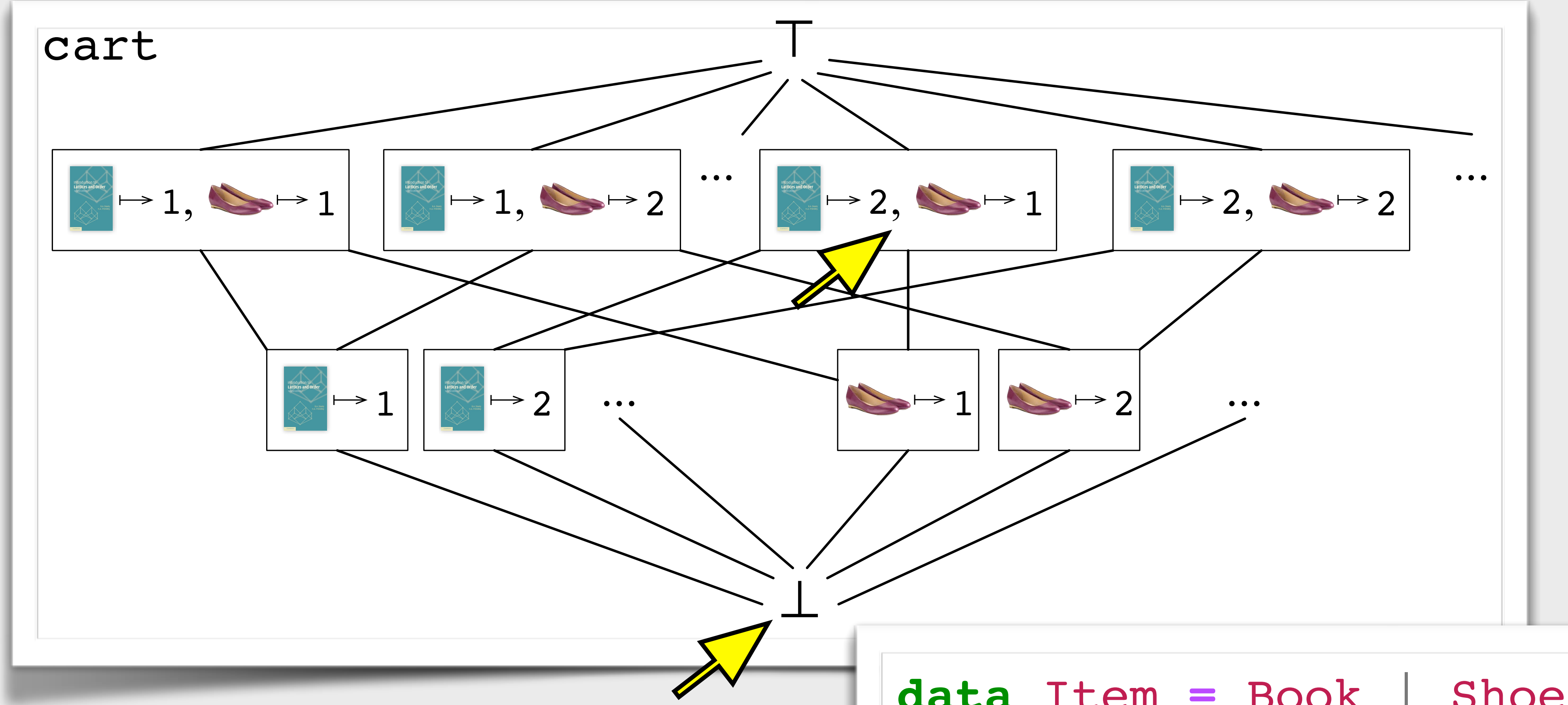
```
data Item = Book | Shoes | ...

p = do
  cart <- newEmptyMap
  fork (insert Shoes 1 cart)
  fork (insert Book 2 cart)
  getKey Book cart -- returns 2
```



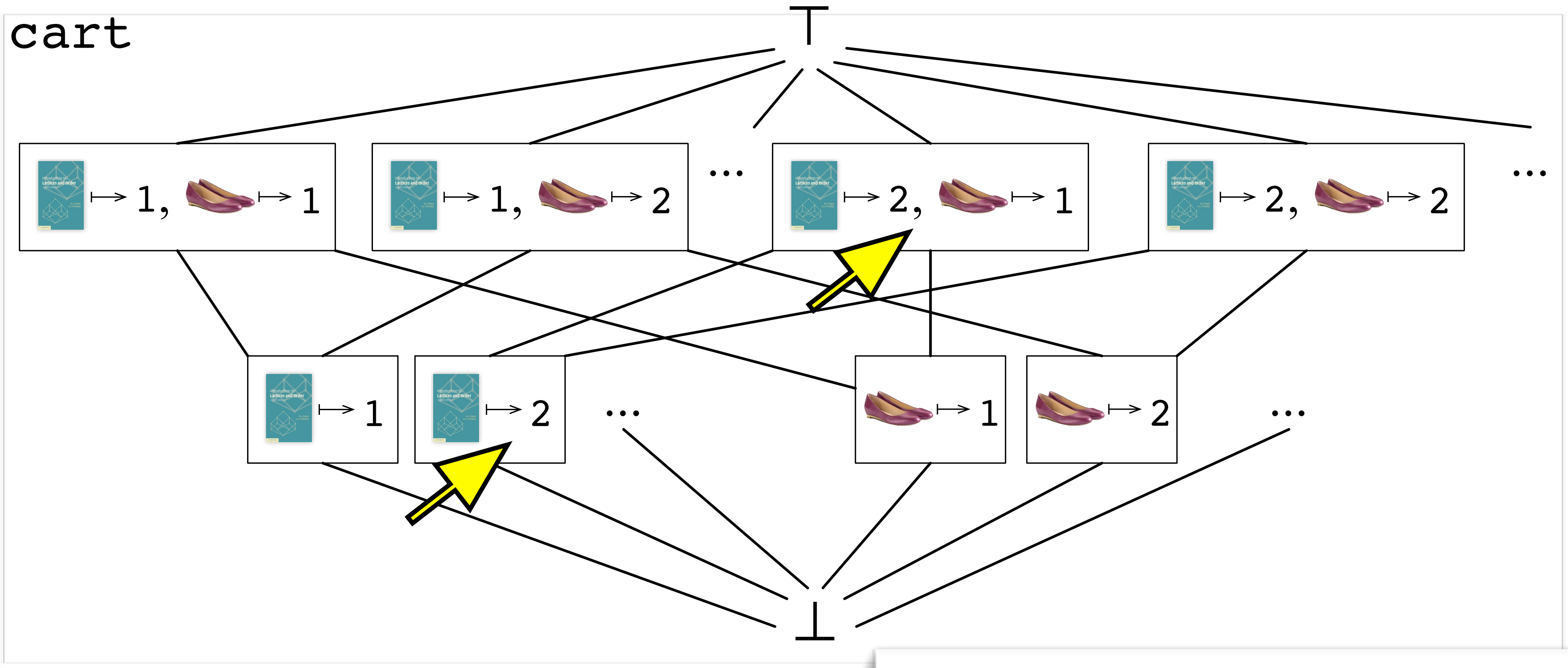
```
data Item = Book | Shoes | ...

p = do
  cart <- newEmptyMap
  fork (insert Shoes 1 cart)
  fork (insert Book 2 cart)
  getKey Book cart -- returns 2
```



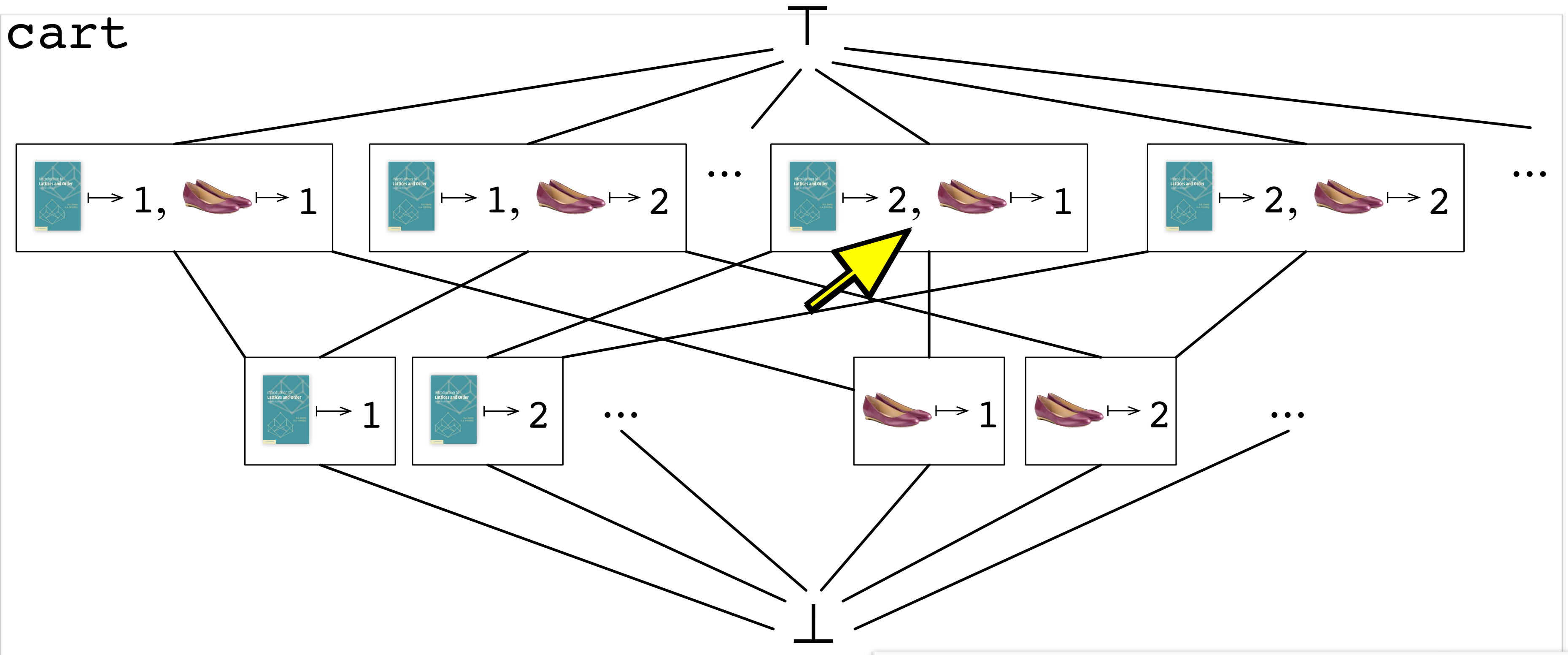
```
data Item = Book | Shoes | ...

p = do
  cart <- newEmptyMap
  fork (insert Shoes 1 cart)
  fork (insert Book 2 cart)
  getKey Book cart -- returns 2
```



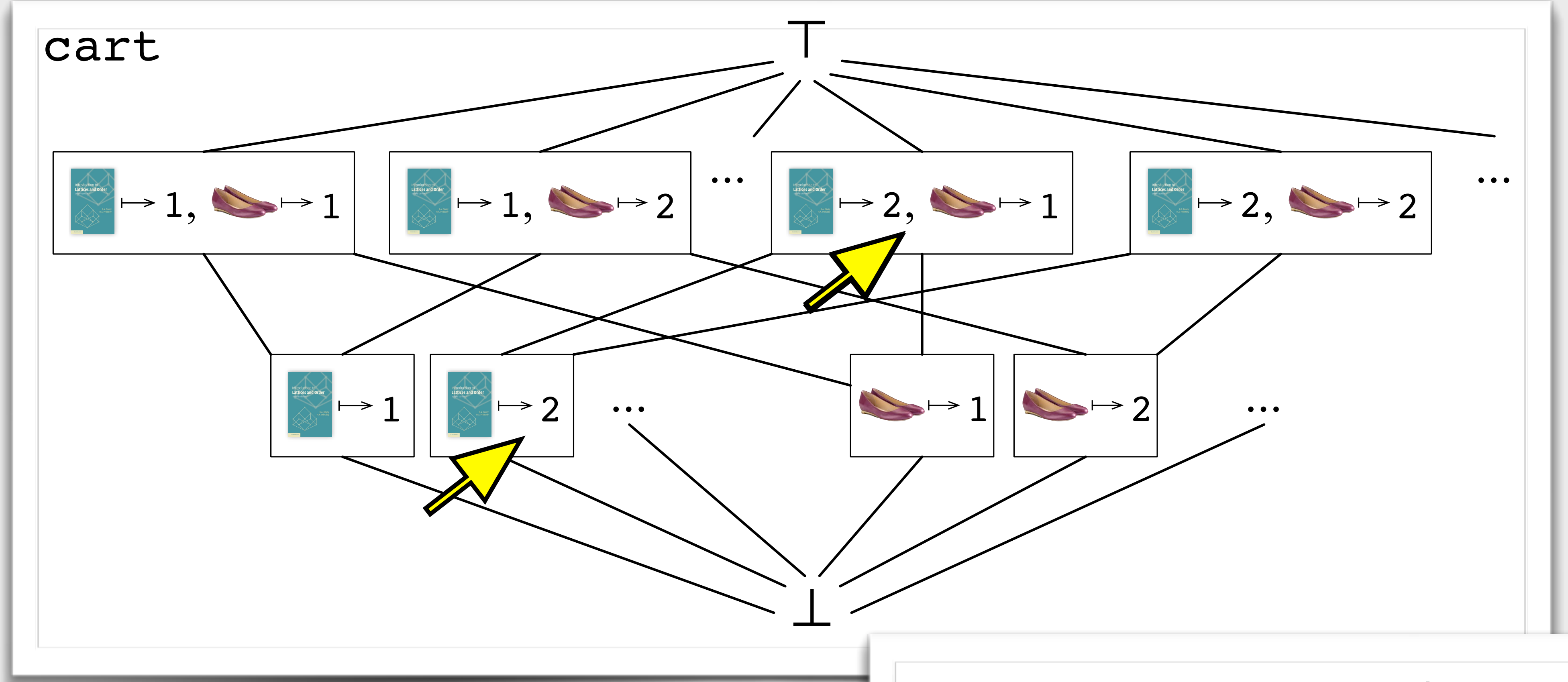
```
data Item = Book | Shoes | ...

p = do
  cart <- newEmptyMap
  fork (insert Shoes 1 cart)
  fork (insert Book 2 cart)
  getKey Book cart -- returns 2
```



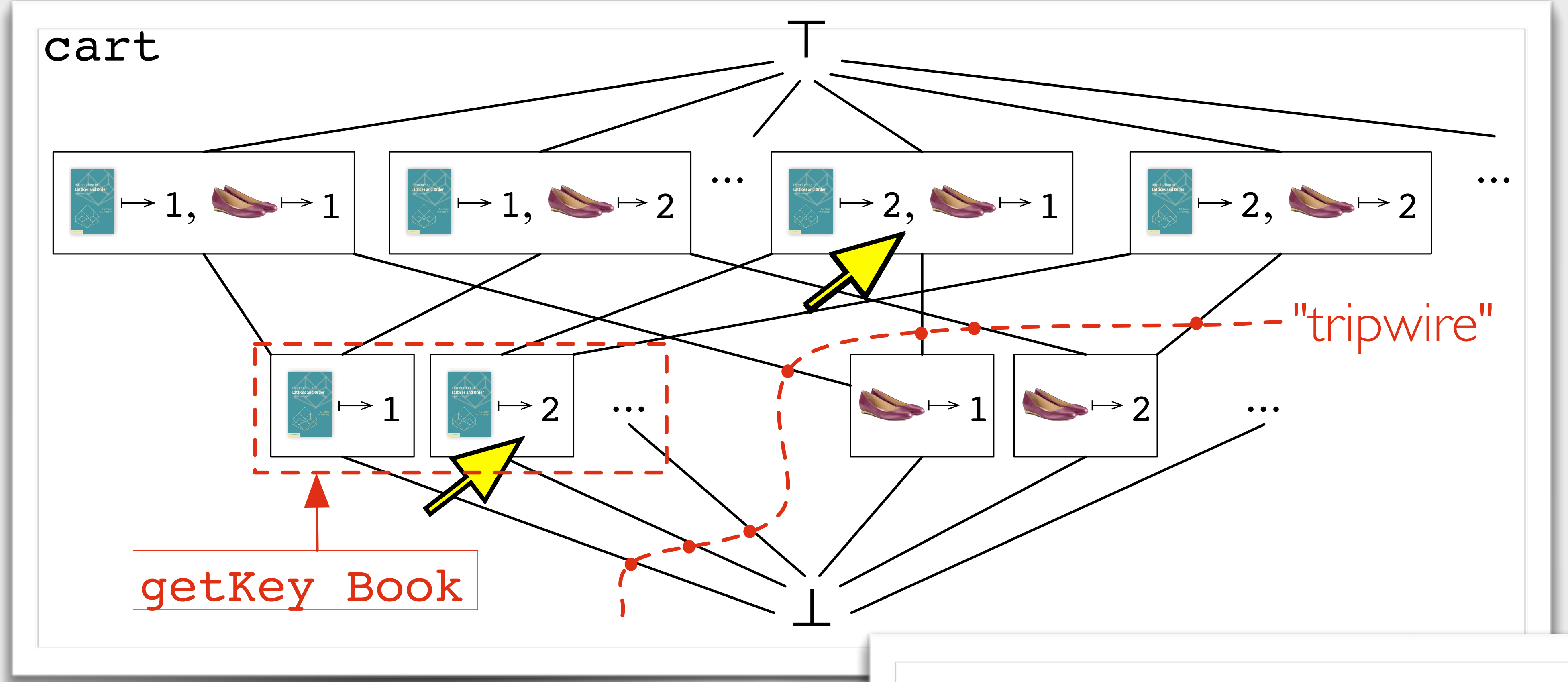
```
data Item = Book | Shoes | ...

p = do
  cart <- newEmptyMap
  fork (insert Shoes 1 cart)
  fork (insert Book 2 cart)
  getKey Book cart -- returns 2
```



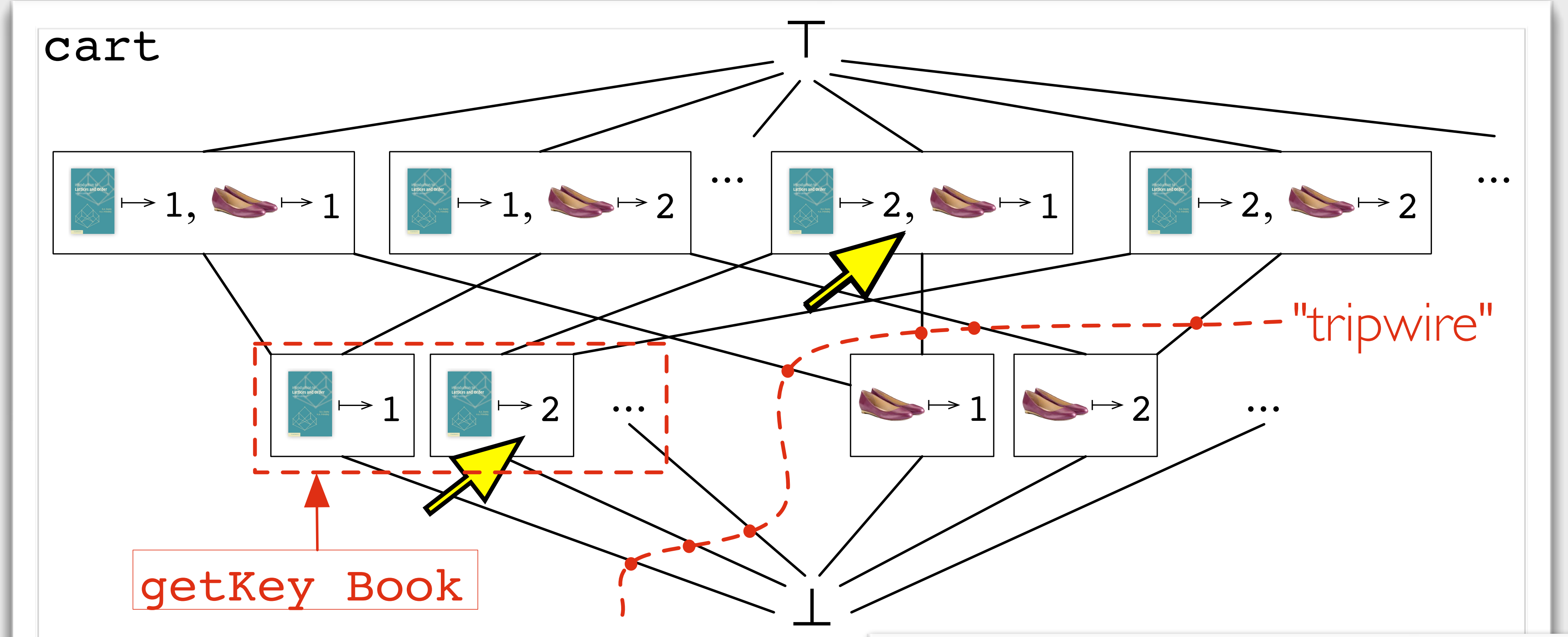
```
data Item = Book | Shoes | ...

p = do
  cart <- newEmptyMap
  fork (insert Shoes 1 cart)
  fork (insert Book 2 cart)
  getKey Book cart -- returns 2
```



```
data Item = Book | Shoes | ...

p = do
  cart <- newEmptyMap
  fork (insert Shoes 1 cart)
  fork (insert Book 2 cart)
  getKey Book cart -- returns 2
```

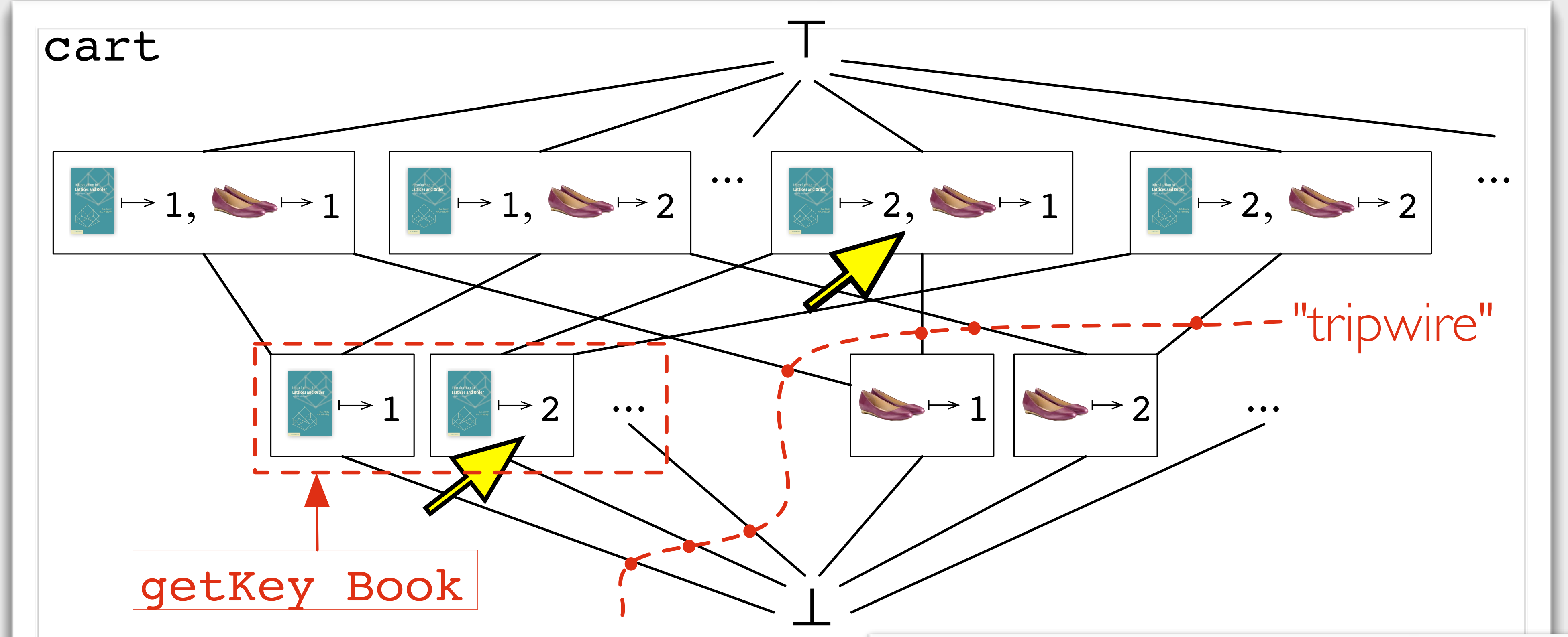


```
data Item = Book | Shoes | ...

p = do
  cart <- newEmptyMap
  fork (insert Shoes 1 cart)
  fork (insert Book 2 cart)
  getKey Book cart -- returns 2
```



```
data Item = Book | Shoes | ...

p = do
  cart <- newEmptyMap
  fork (insert Shoes 1 cart)
  fork (insert Book 2 cart)
  getKey Book cart -- returns 2
```

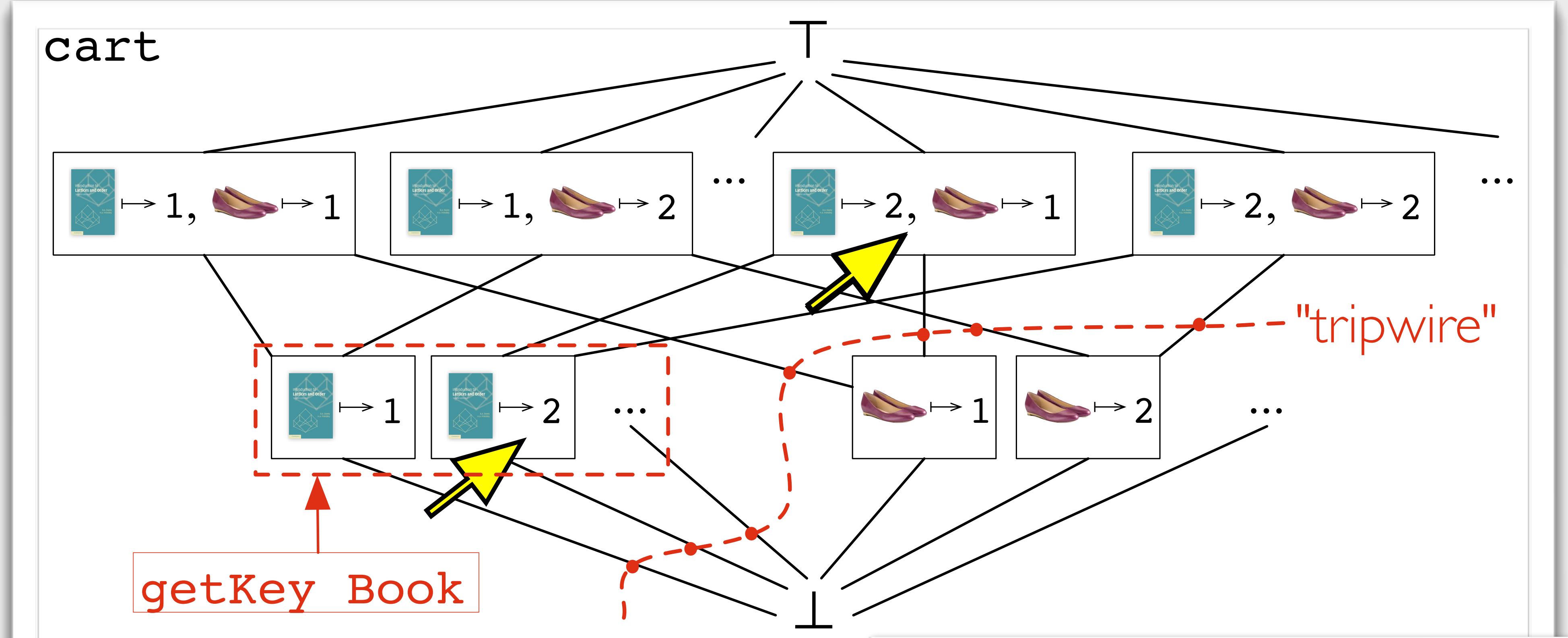
- ☑ Writes must be commutative and inflationary
- ☑ Threshold sets must be *pairwise incompatible*

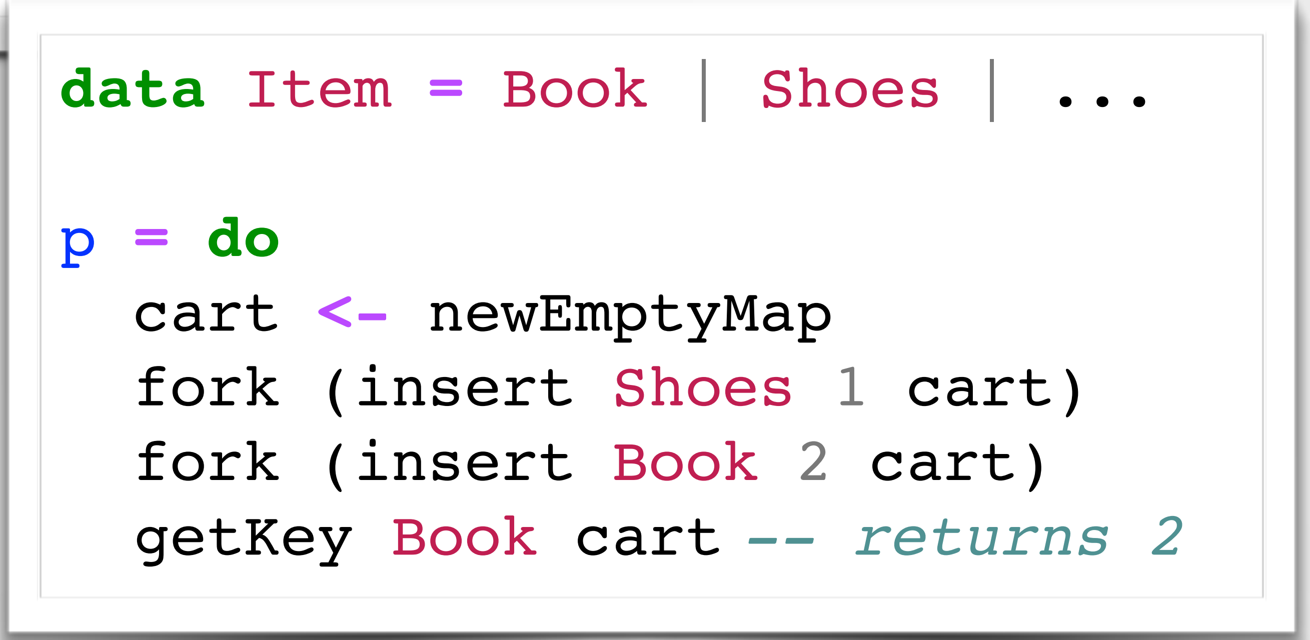


```
data Item = Book | Shoes | ...

p = do
  cart <- newEmptyMap
  fork (insert Shoes 1 cart)
  fork (insert Book 2 cart)
  getKey Book cart -- returns 2
```

- ☑ Writes must be commutative and inflationary
- ☑ Threshold sets must be *pairwise incompatible*

library obligations



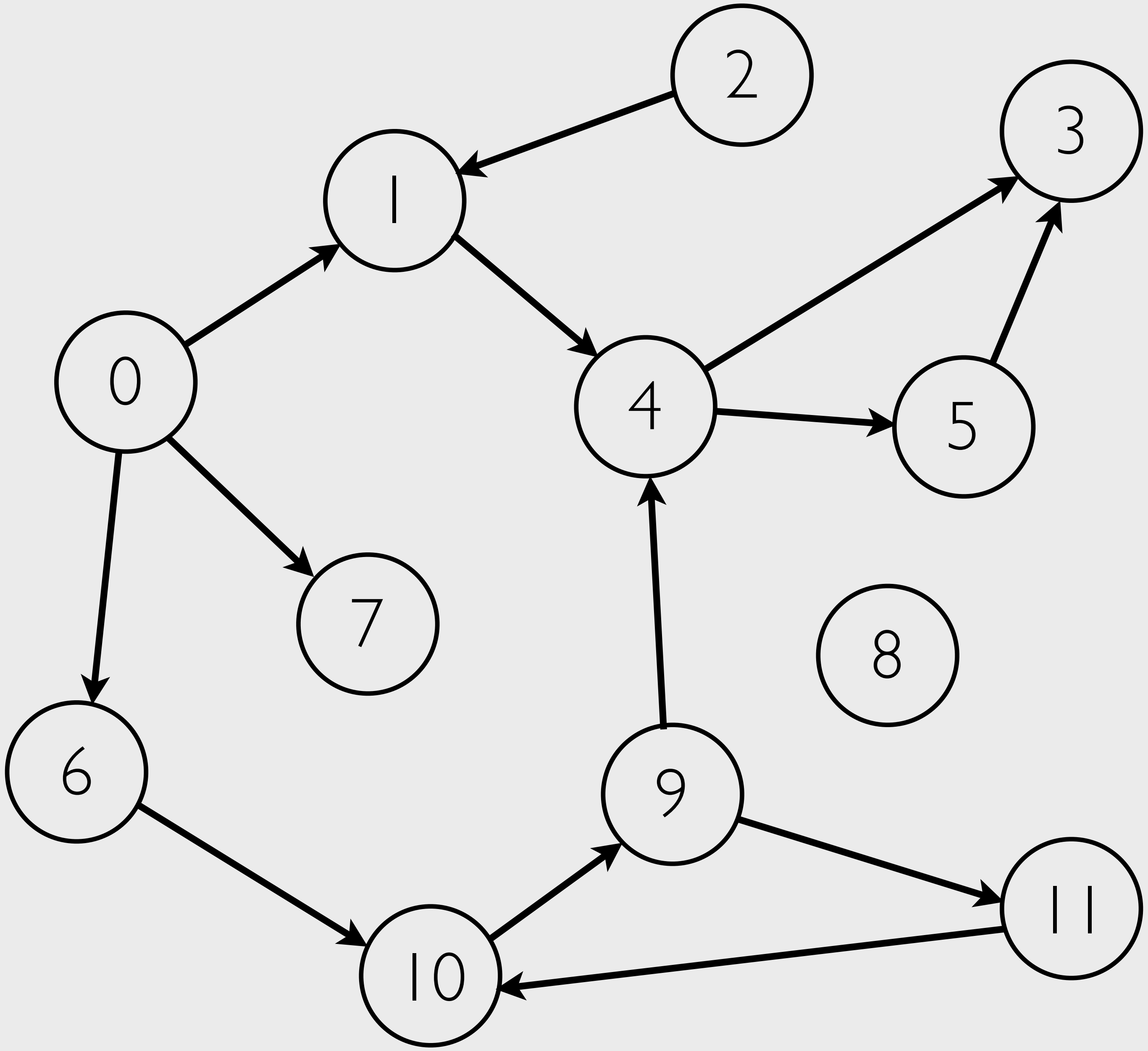


```
data Item = Book | Shoes | ...

p = do
  cart <- newEmptyMap
  fork (insert Shoes 1 cart)
  fork (insert Book 2 cart)
  getKey Book cart -- returns 2
```

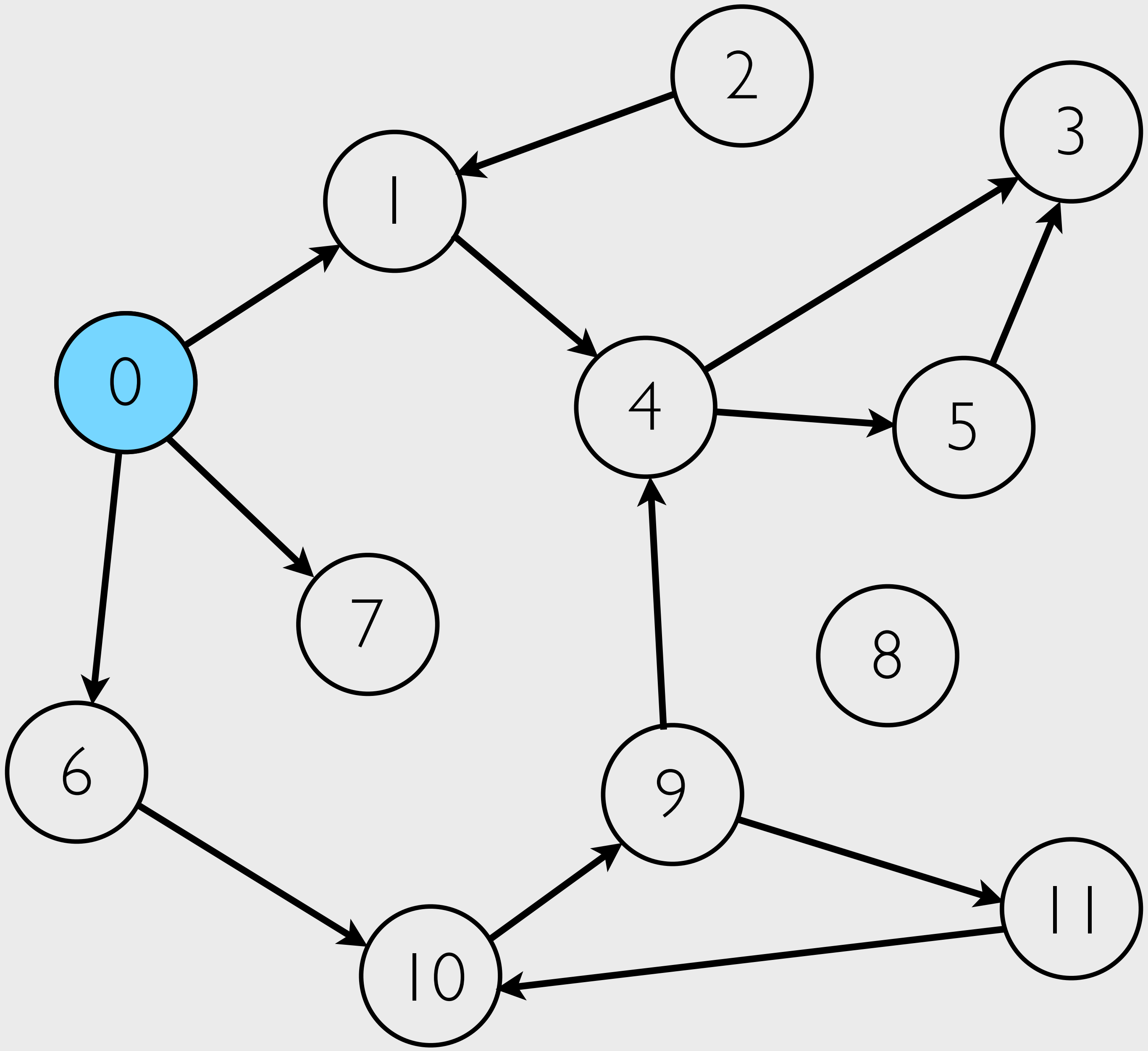
- ☑ Writes must be commutative and inflationary
- ☑ Threshold sets must be *pairwise incompatible*

library obligations

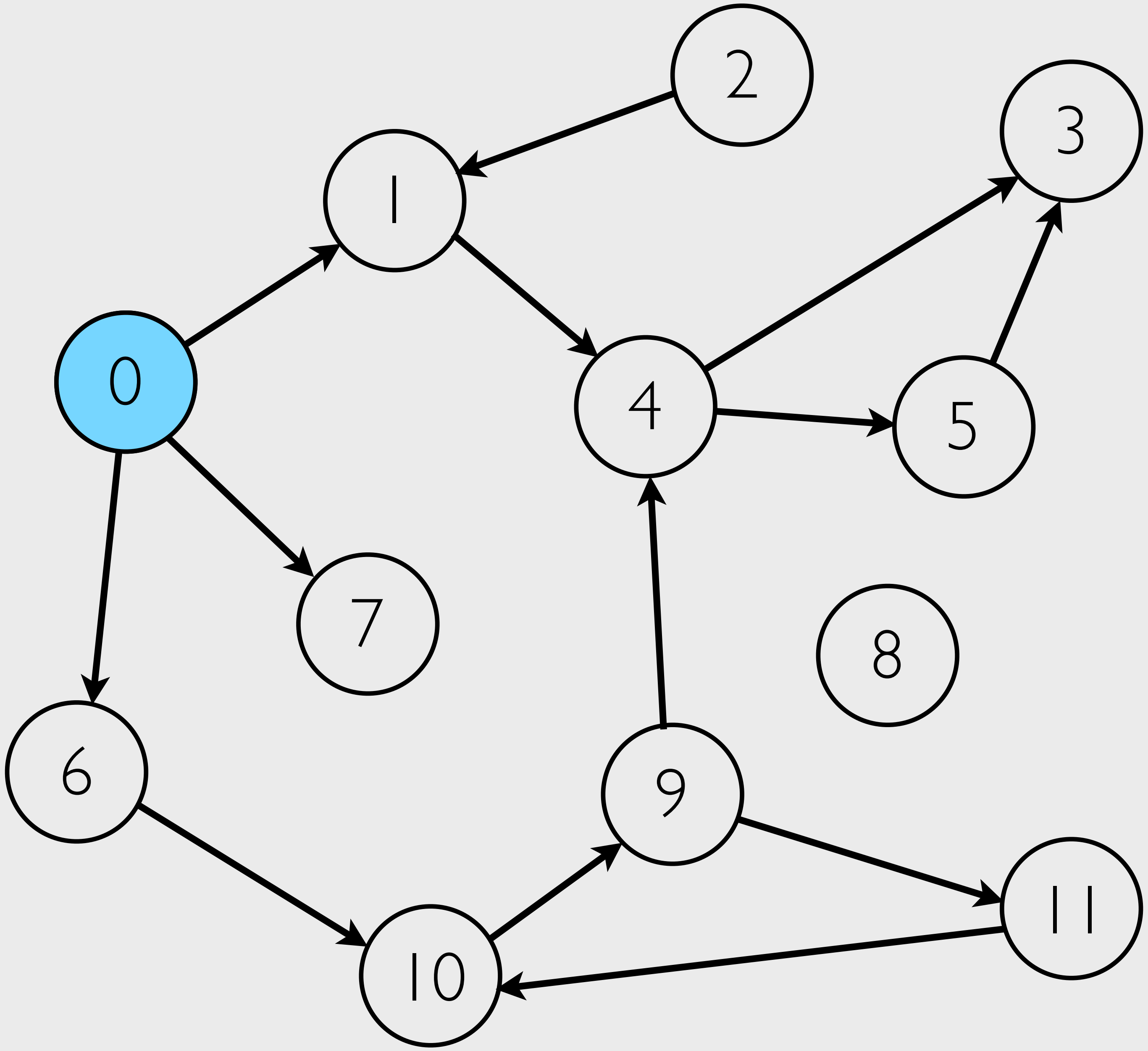
client guarantee

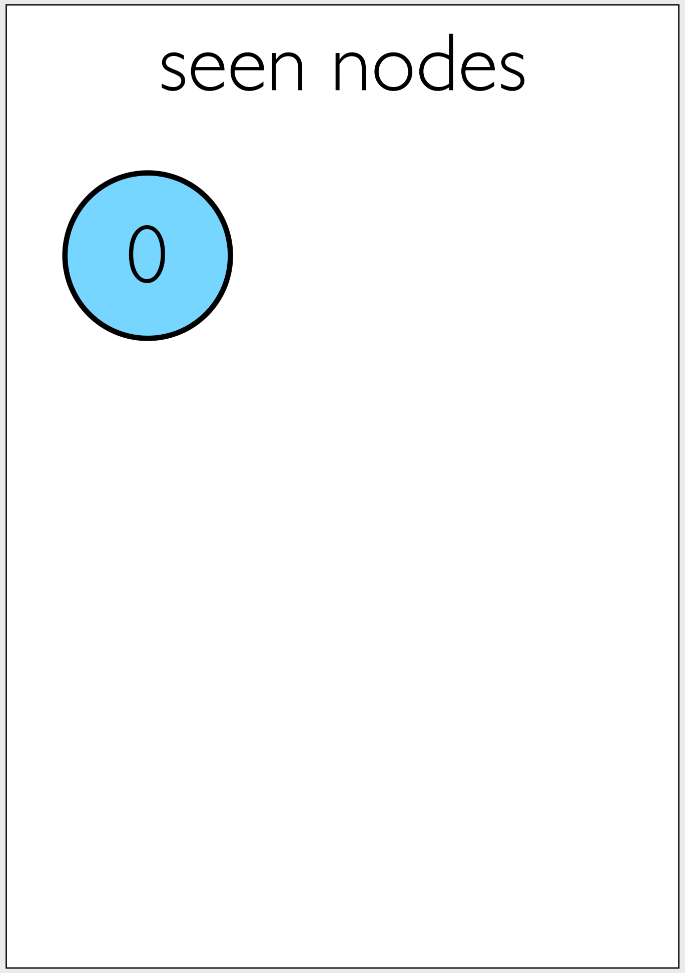


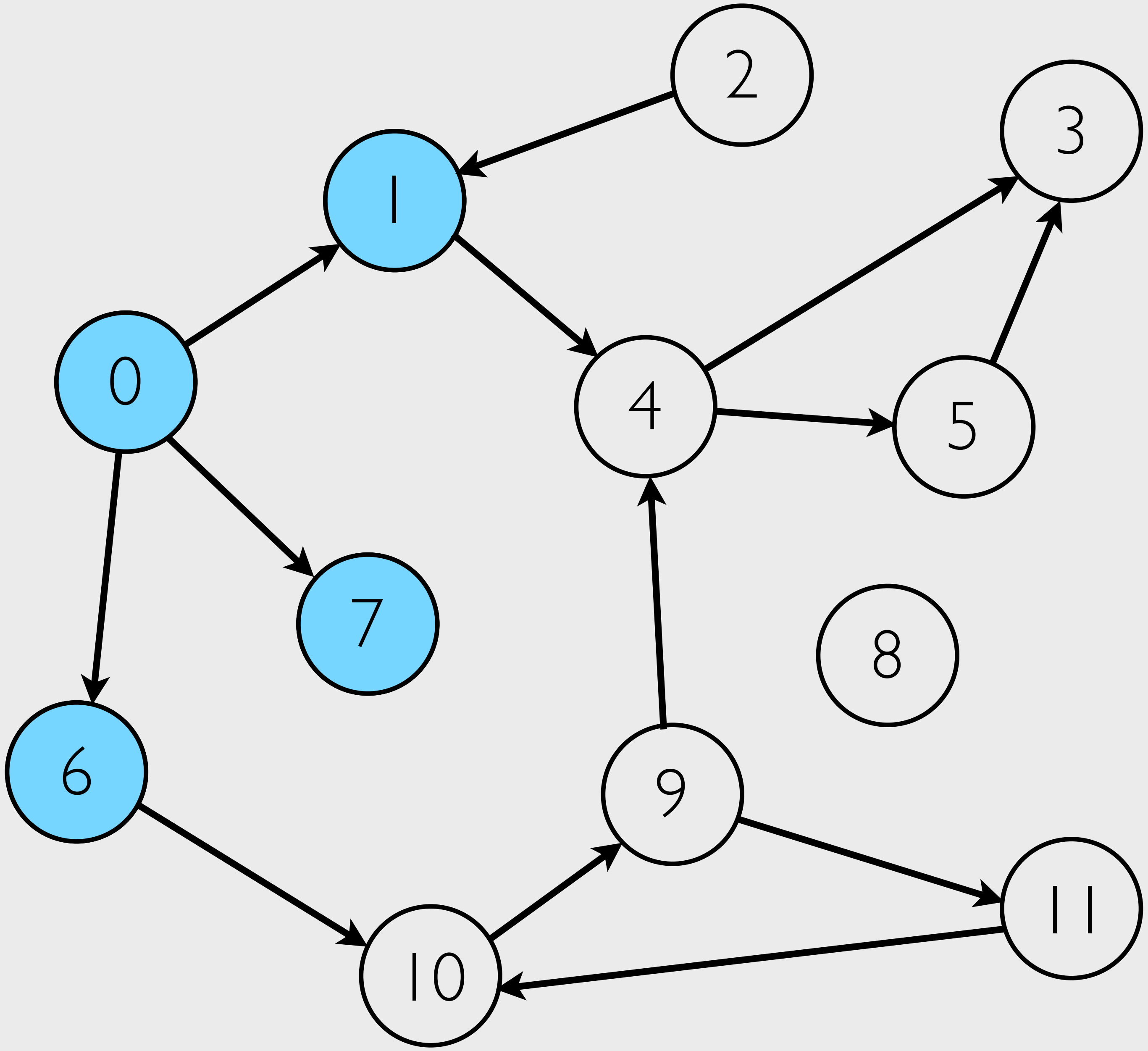
seen nodes

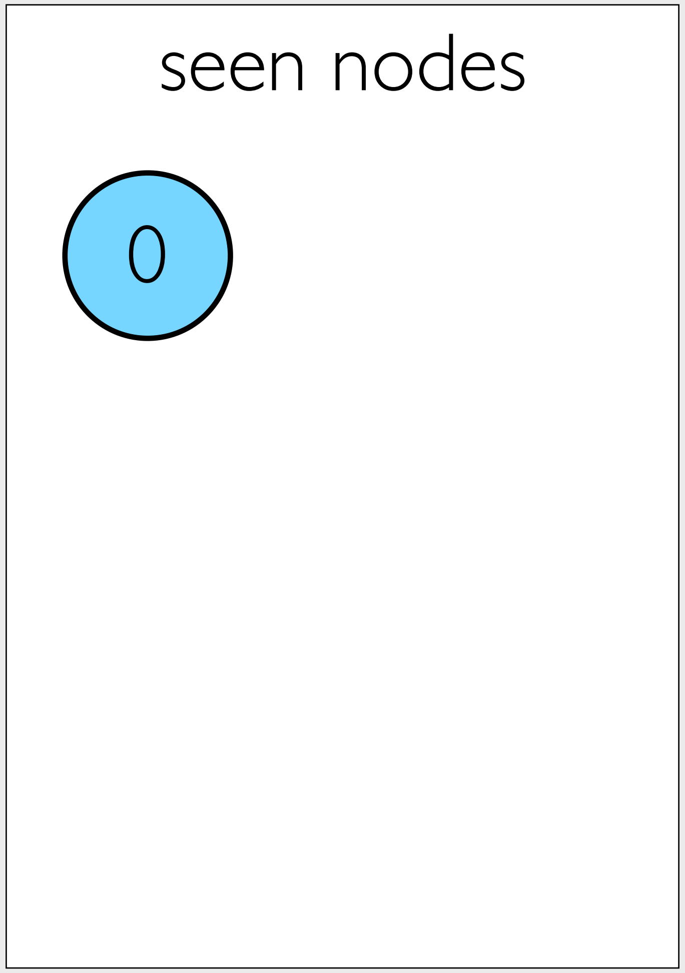


seen nodes

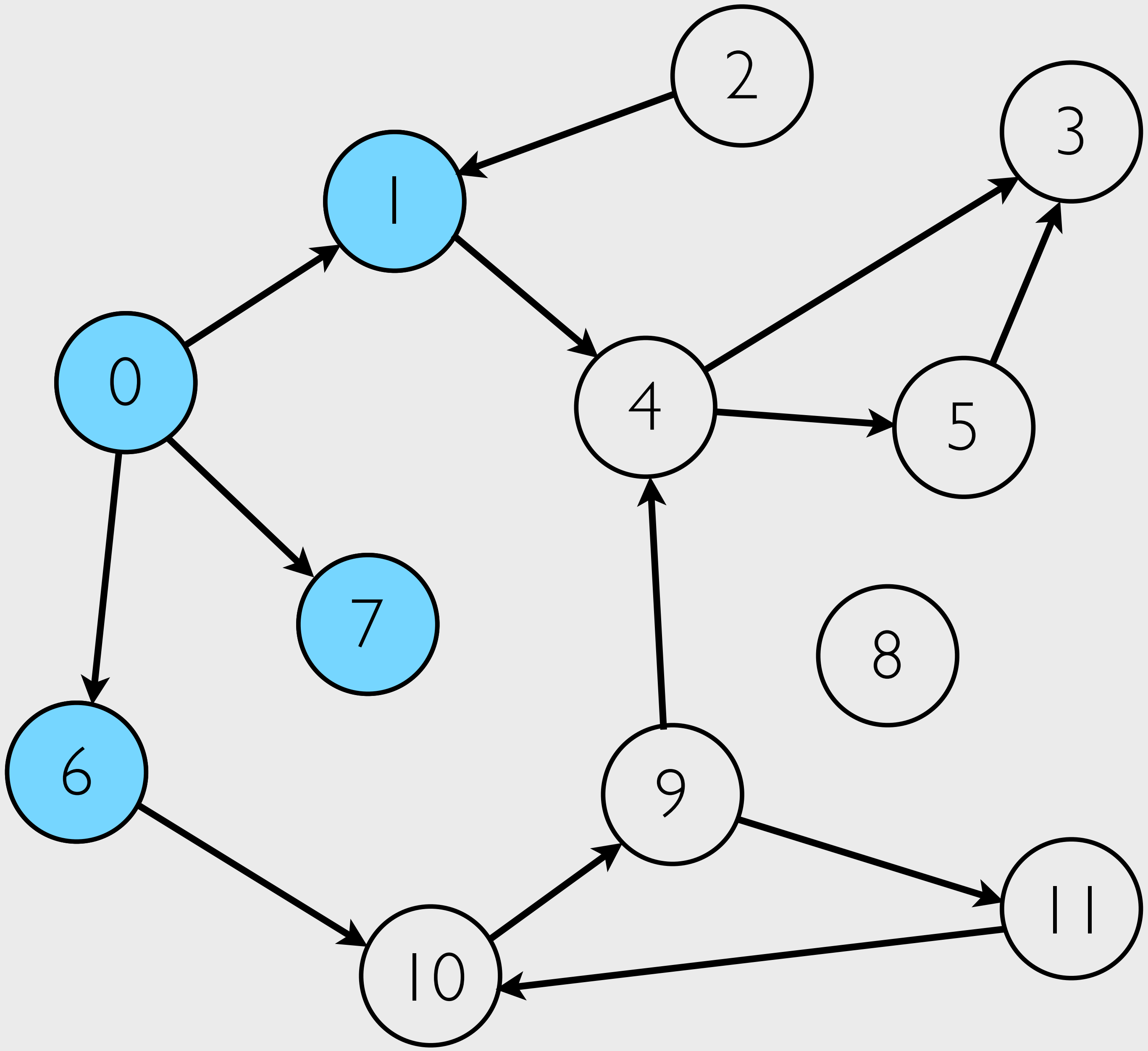


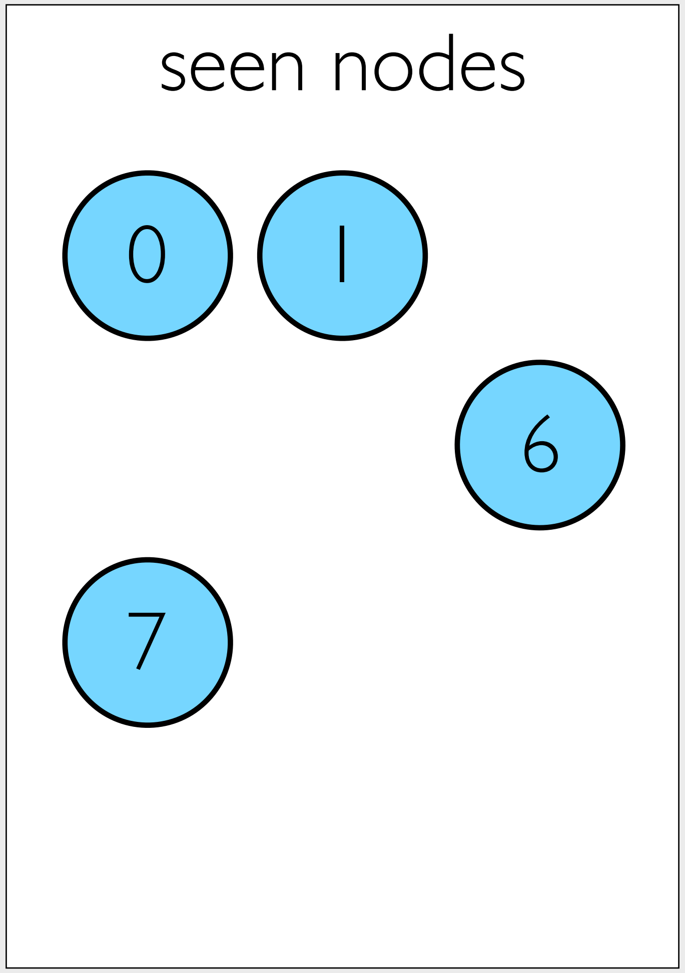

seen nodes

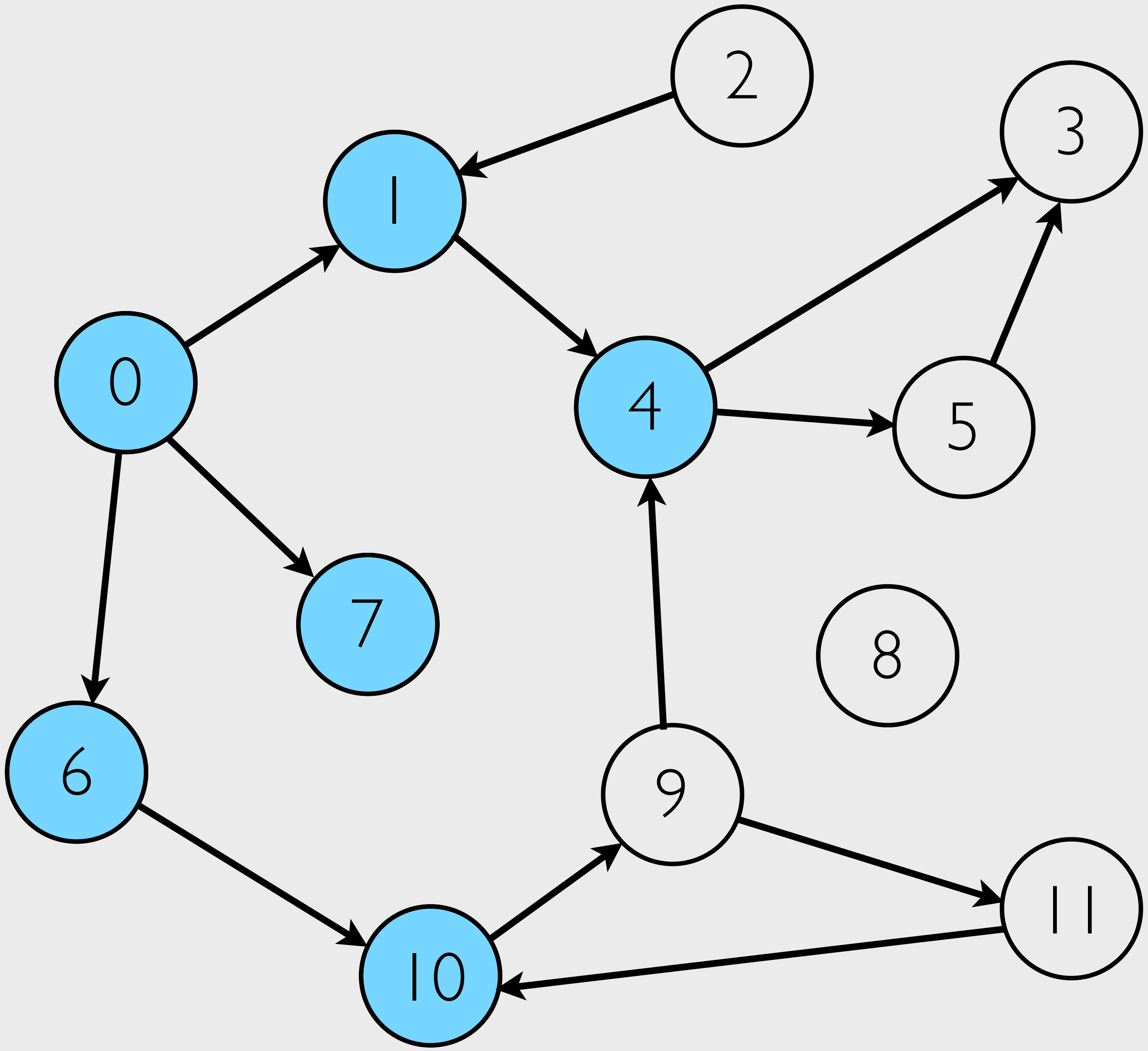


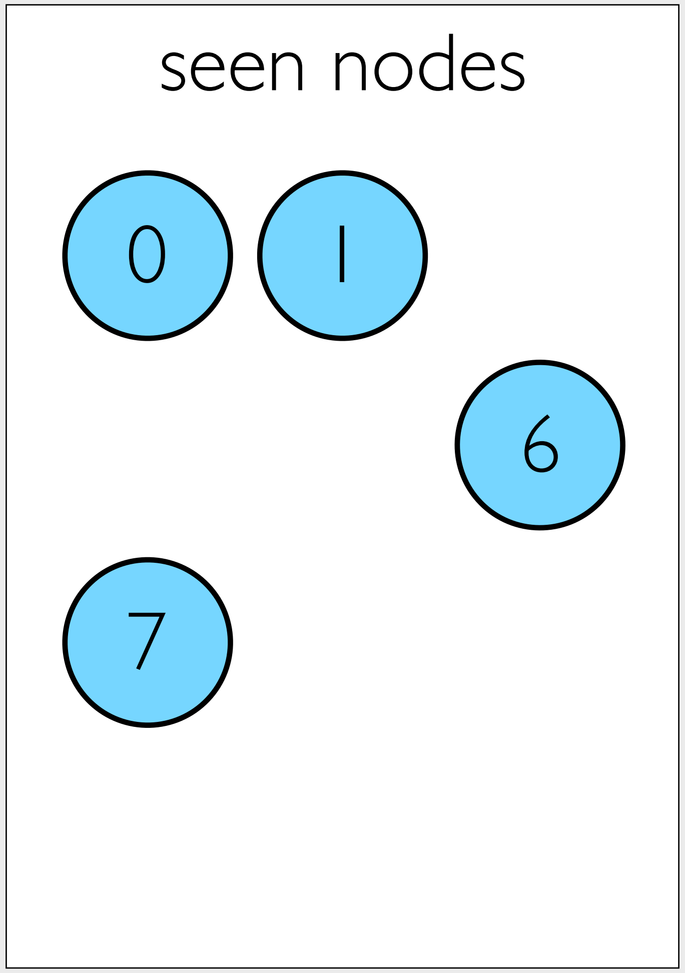


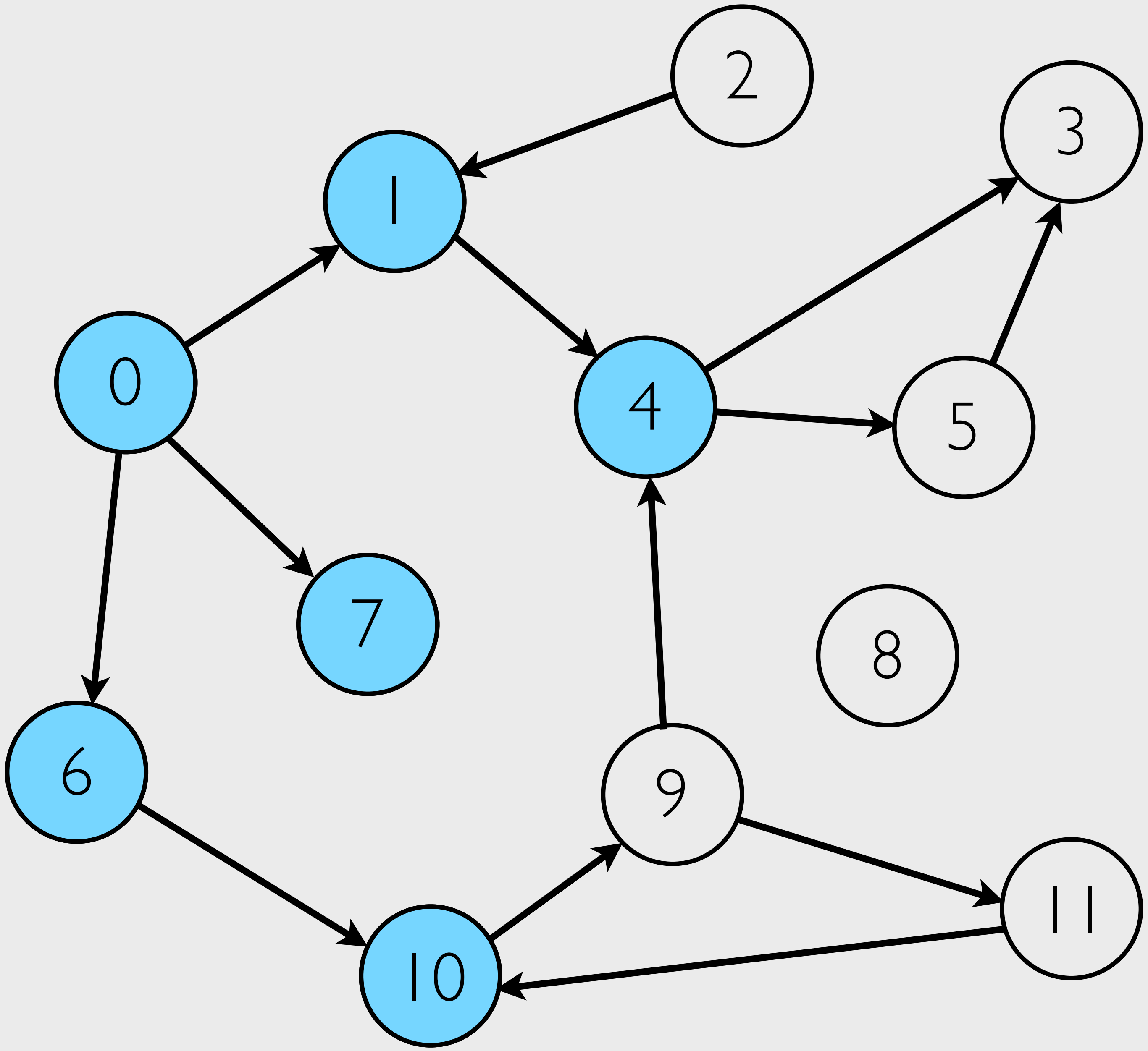
seen nodes

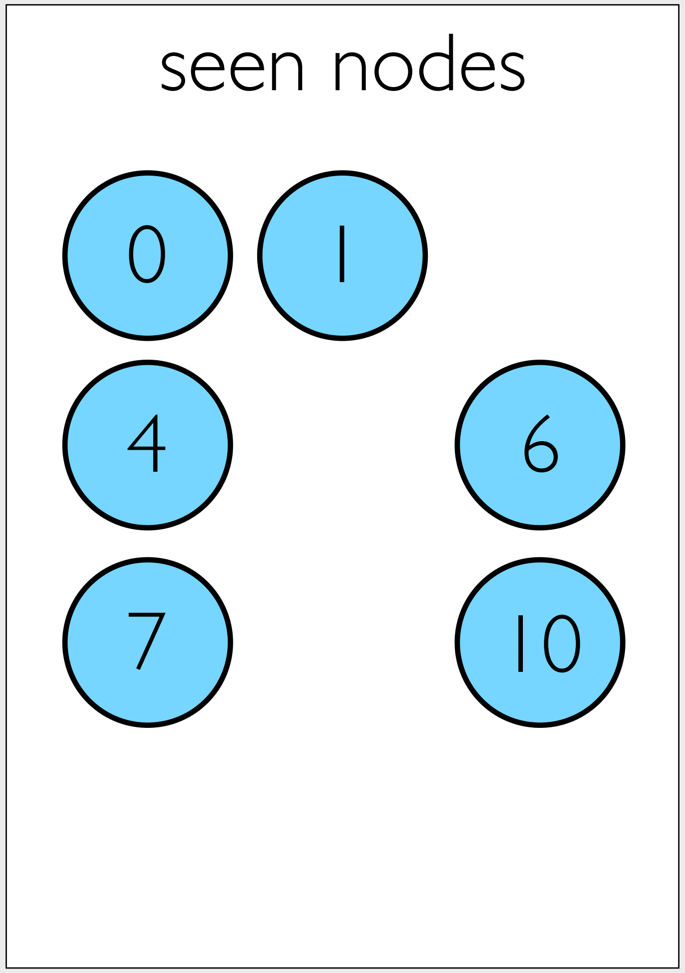

2
3
1
0
4
5
7
8
6
9
11
10

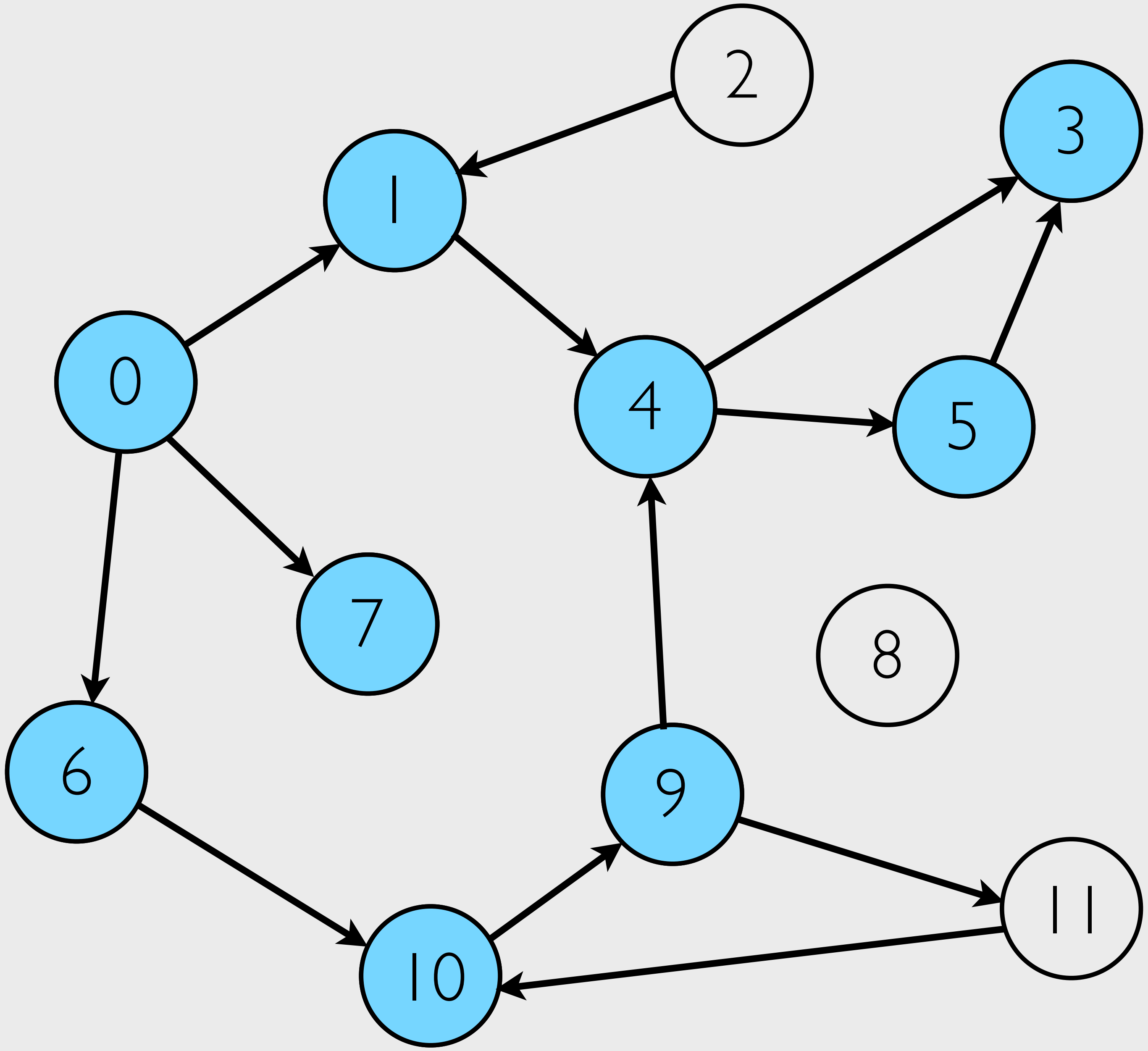

seen nodes
0
1
6
7

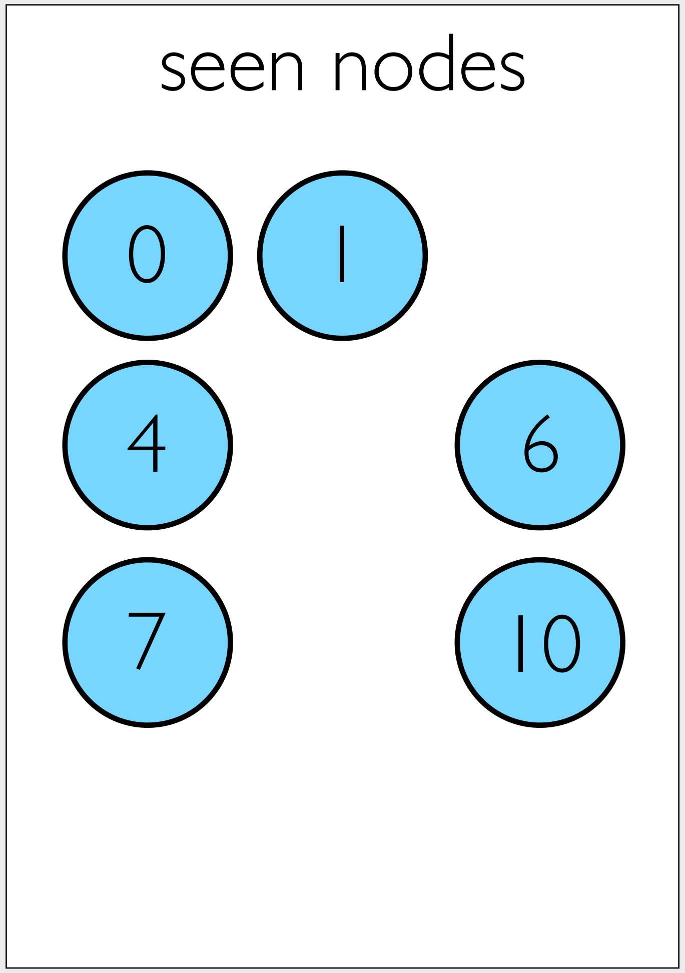

2
1
3
0
4
5
7
8
6
9
10
11

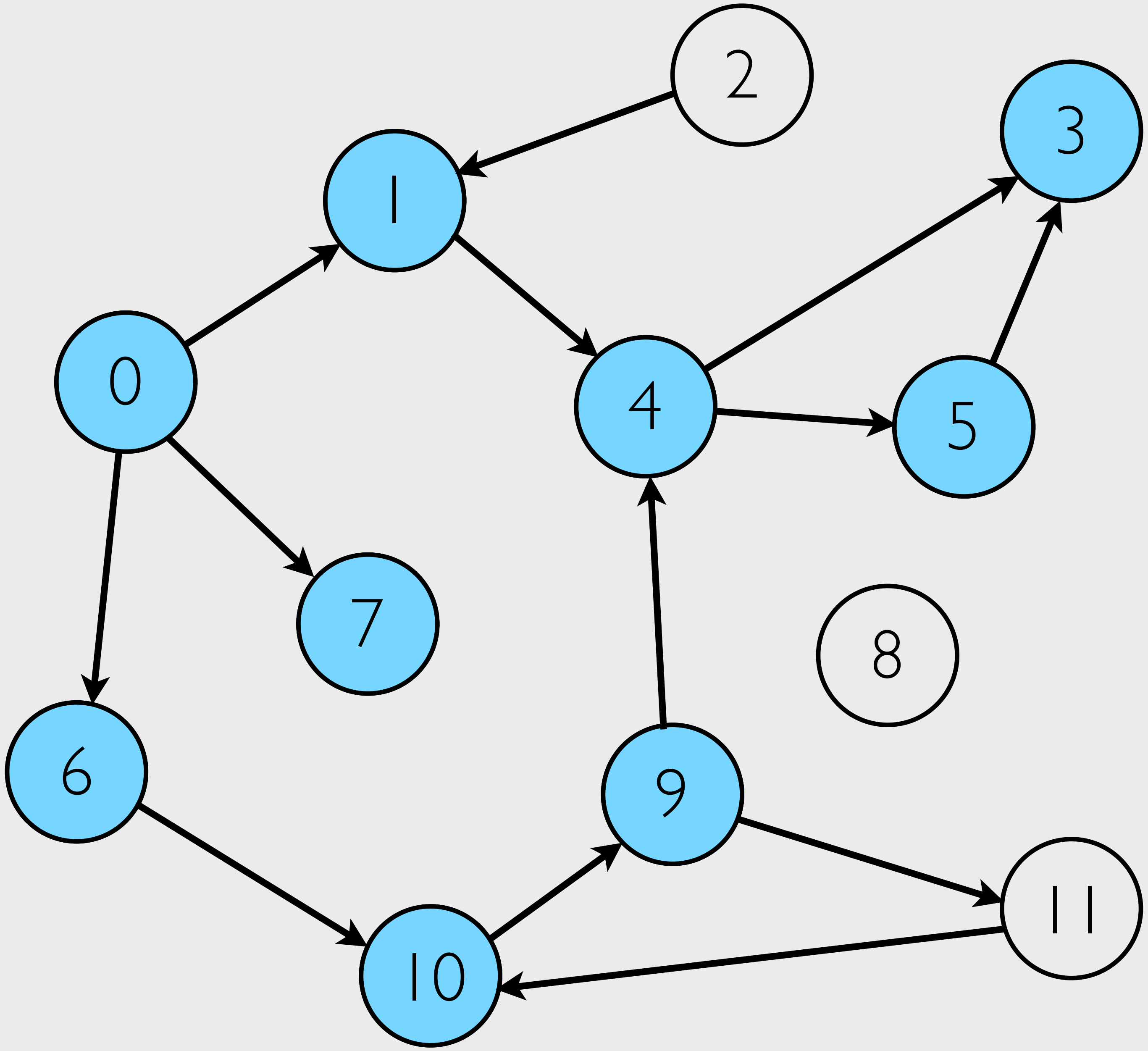

seen nodes
0
1
6
7




seen nodes




seen nodes

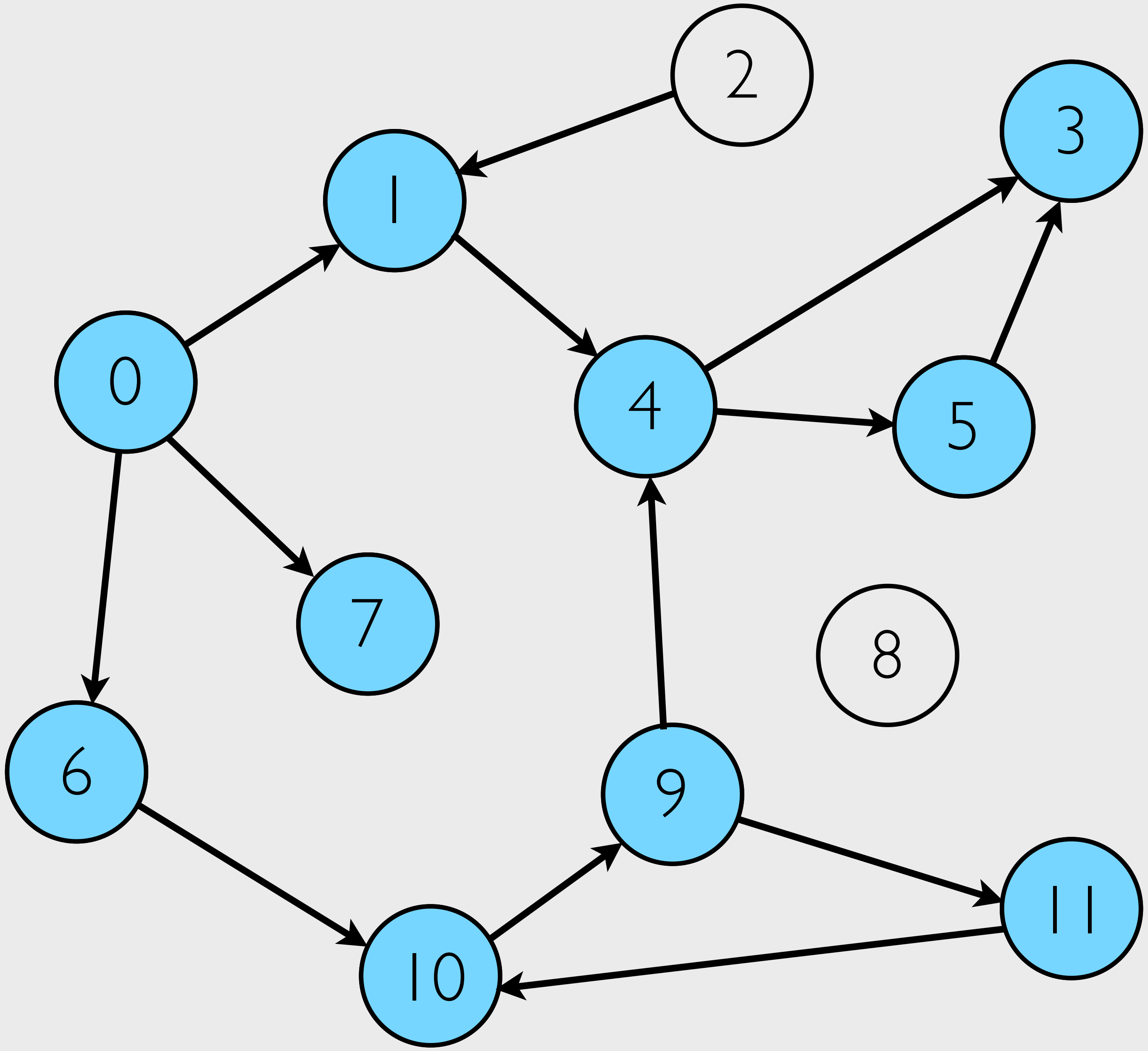


seen nodes

0 1 3

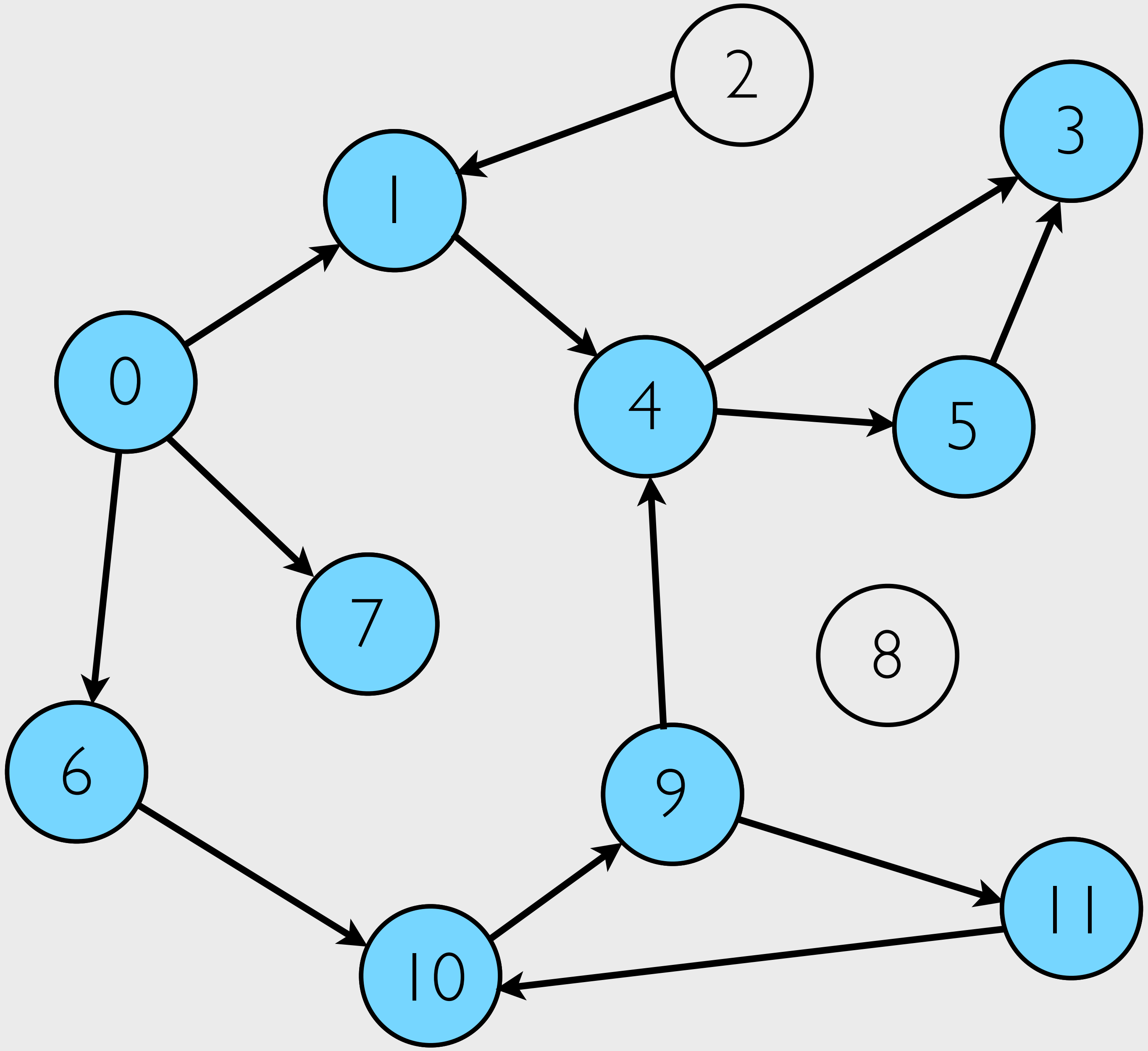
4 5 6

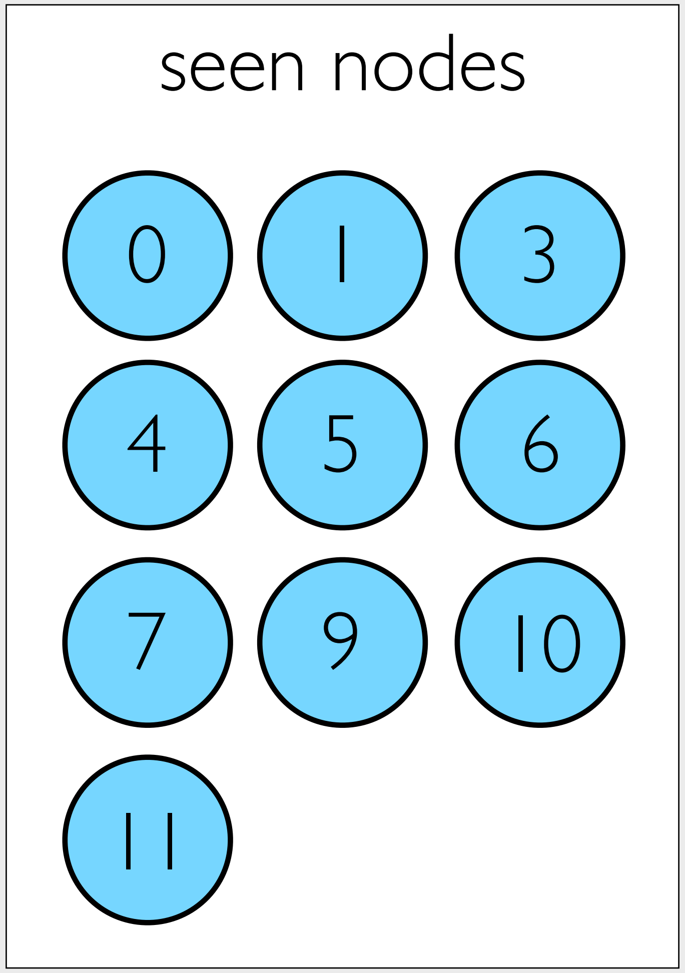
7 9 10



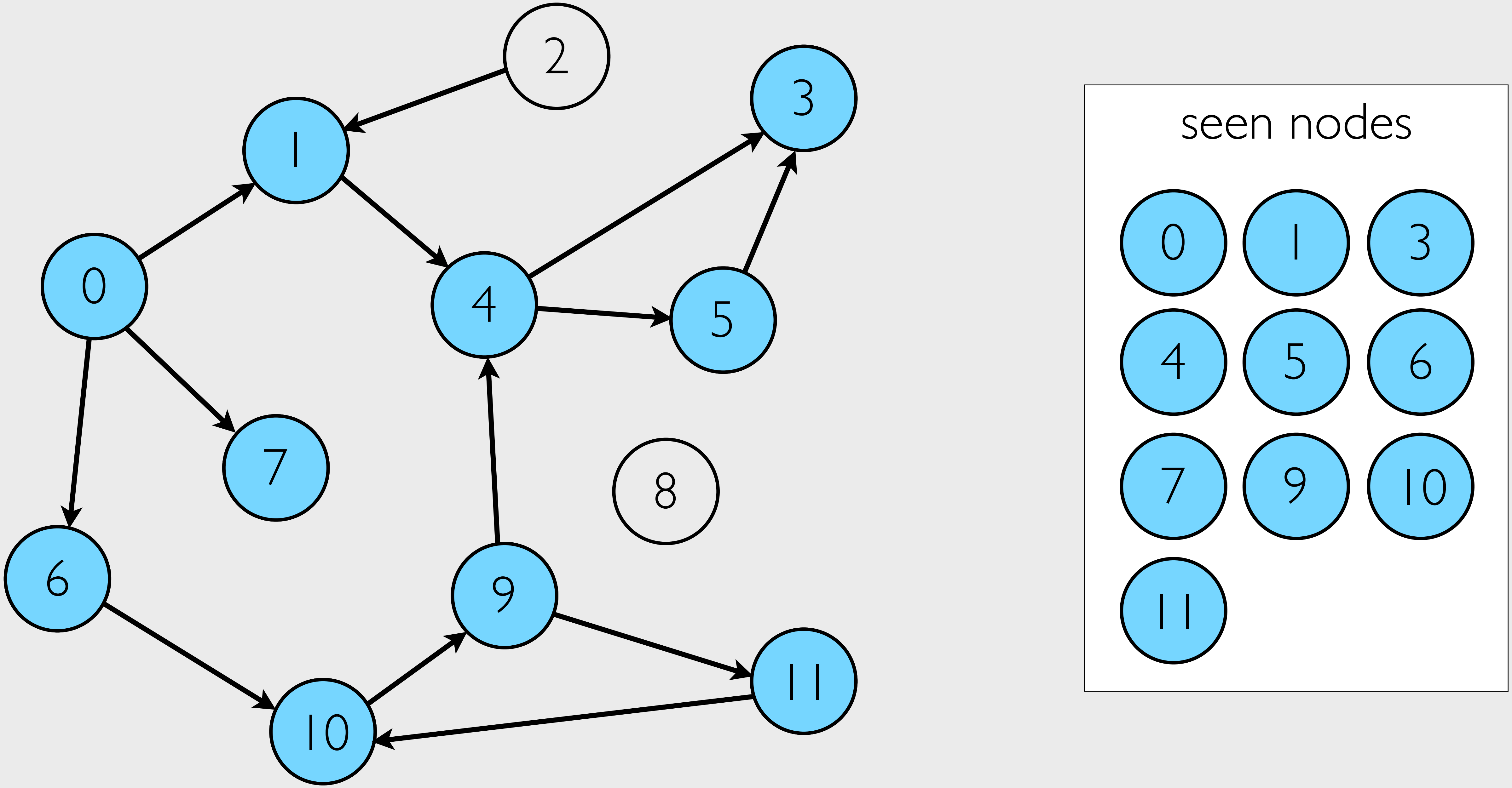
seen nodes

0 1 3
4 5 6
7 9 10




seen nodes

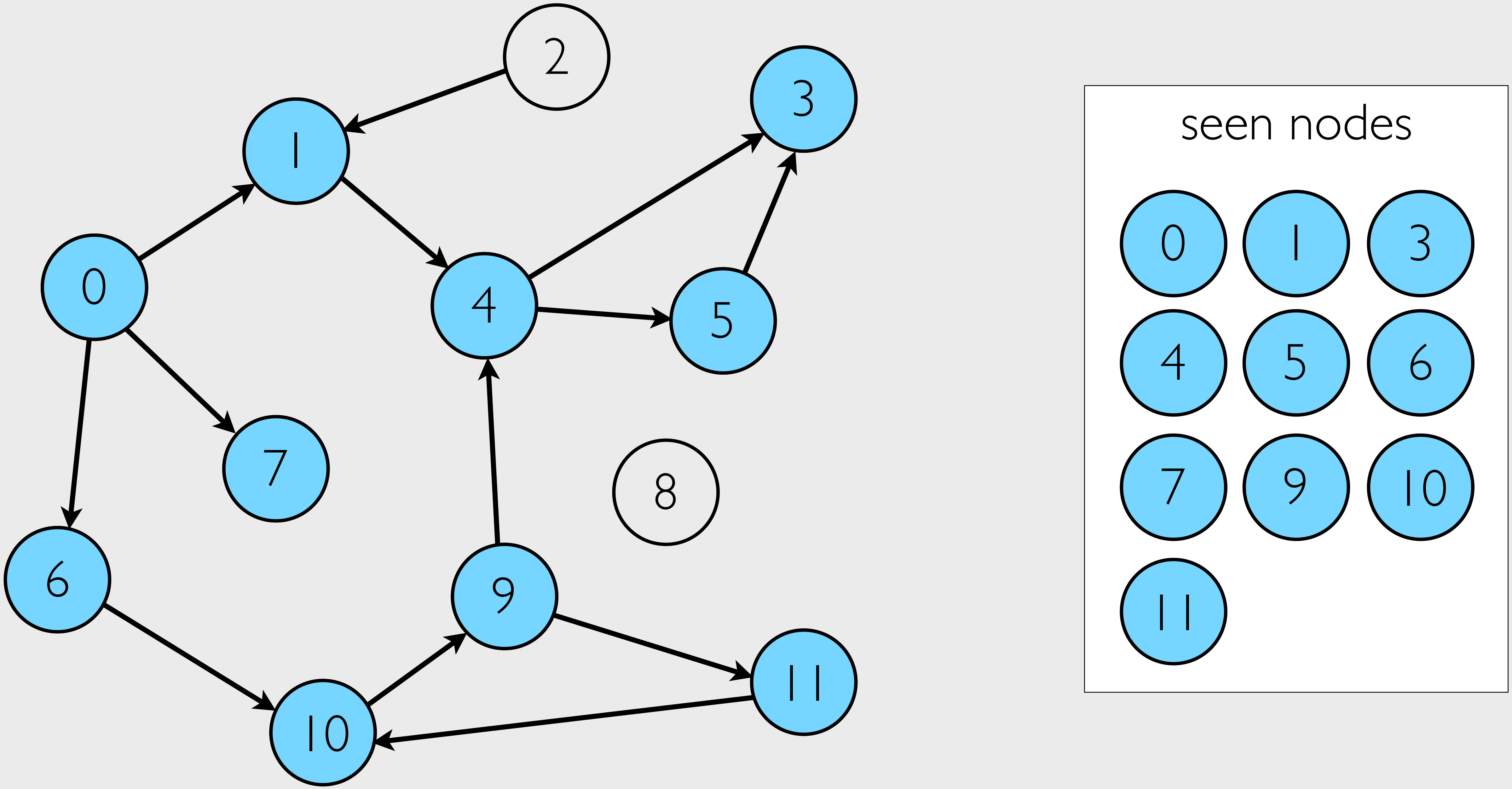
Events are updates that change an LVar's state



seen nodes

Events are updates that change an LVar's state

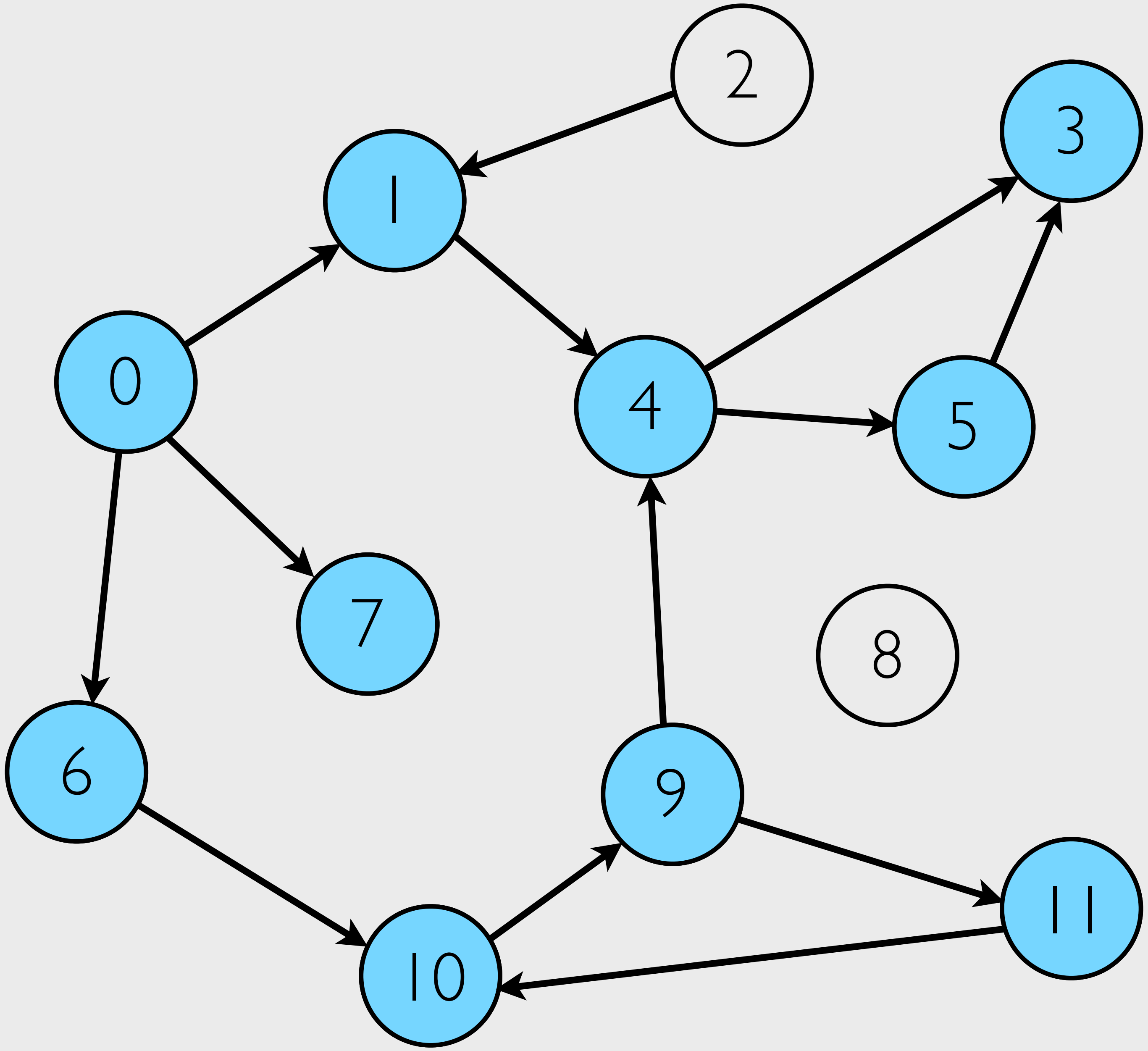
Event handlers listen for events and launch callbacks in response

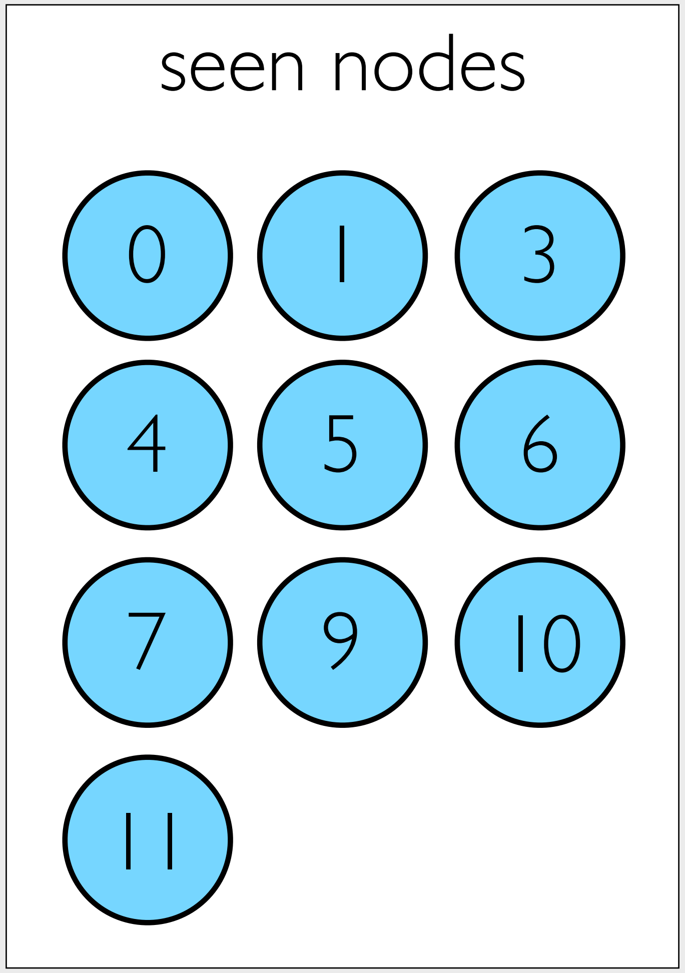


Events are updates that change an LVar's state

Event handlers listen for events and launch callbacks in response

`quiesce` blocks until all callbacks launched by a given handler are done running

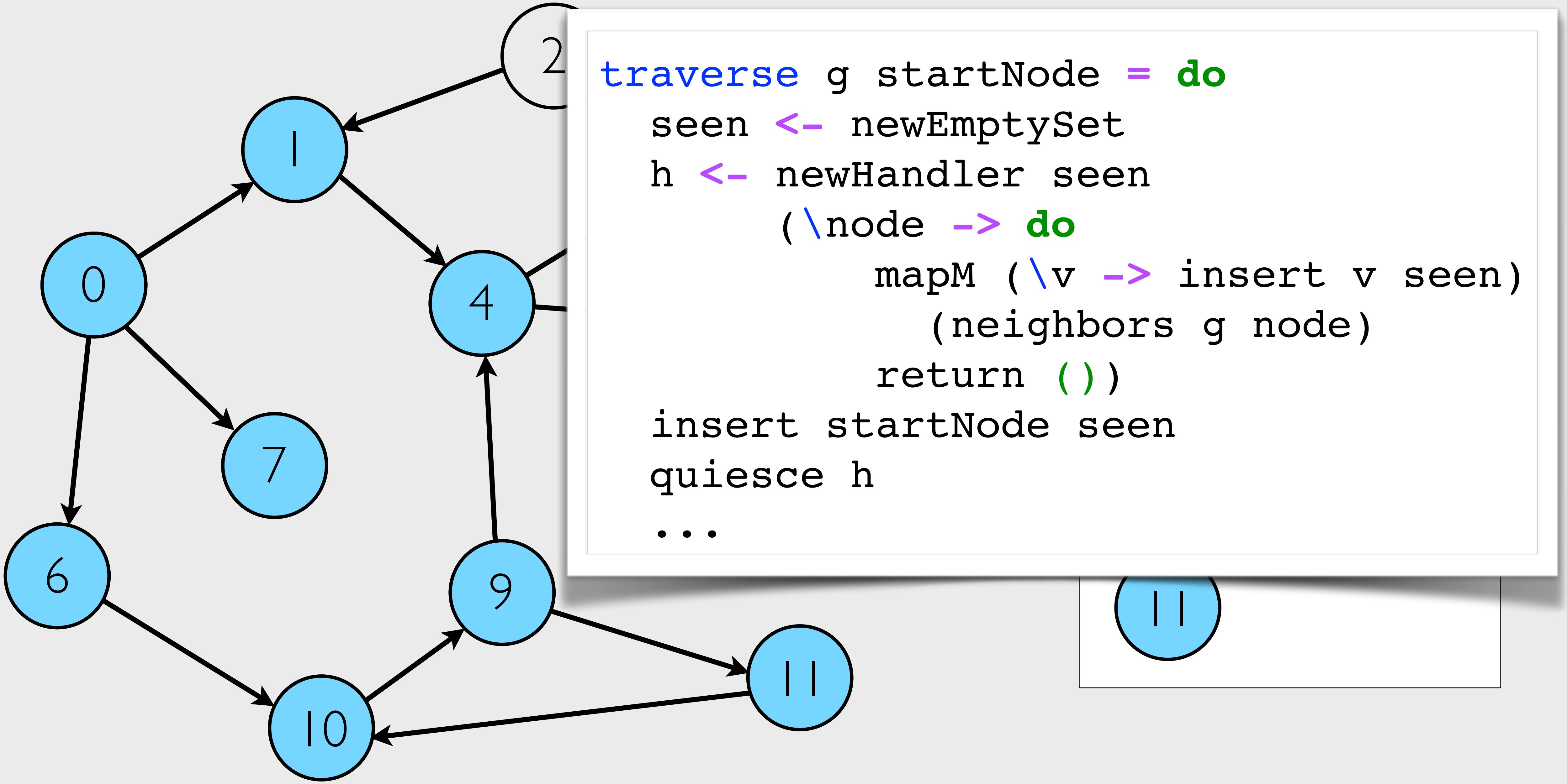




Events are updates that change an LVar's state

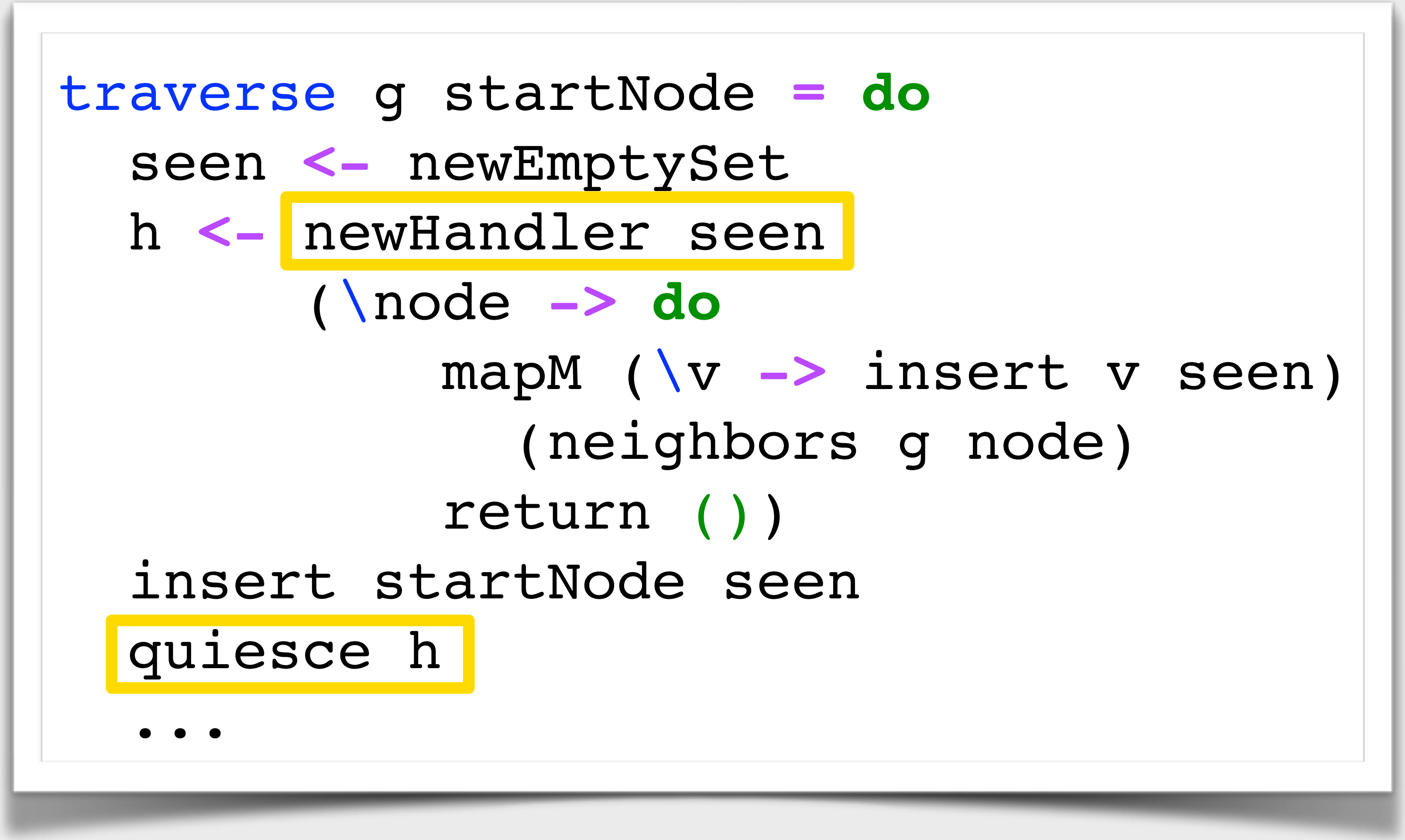
Event handlers listen for events and launch callbacks in response

`quiesce` blocks until all callbacks launched by a given handler are done running



```
traverse g startNode = do
  seen <- newEmptySet
  h <- newHandler seen
       (\node -> do
            mapM (\v -> insert v seen)
              (neighbors g node)
            return ())
  insert startNode seen
  quiesce h
  ...
```

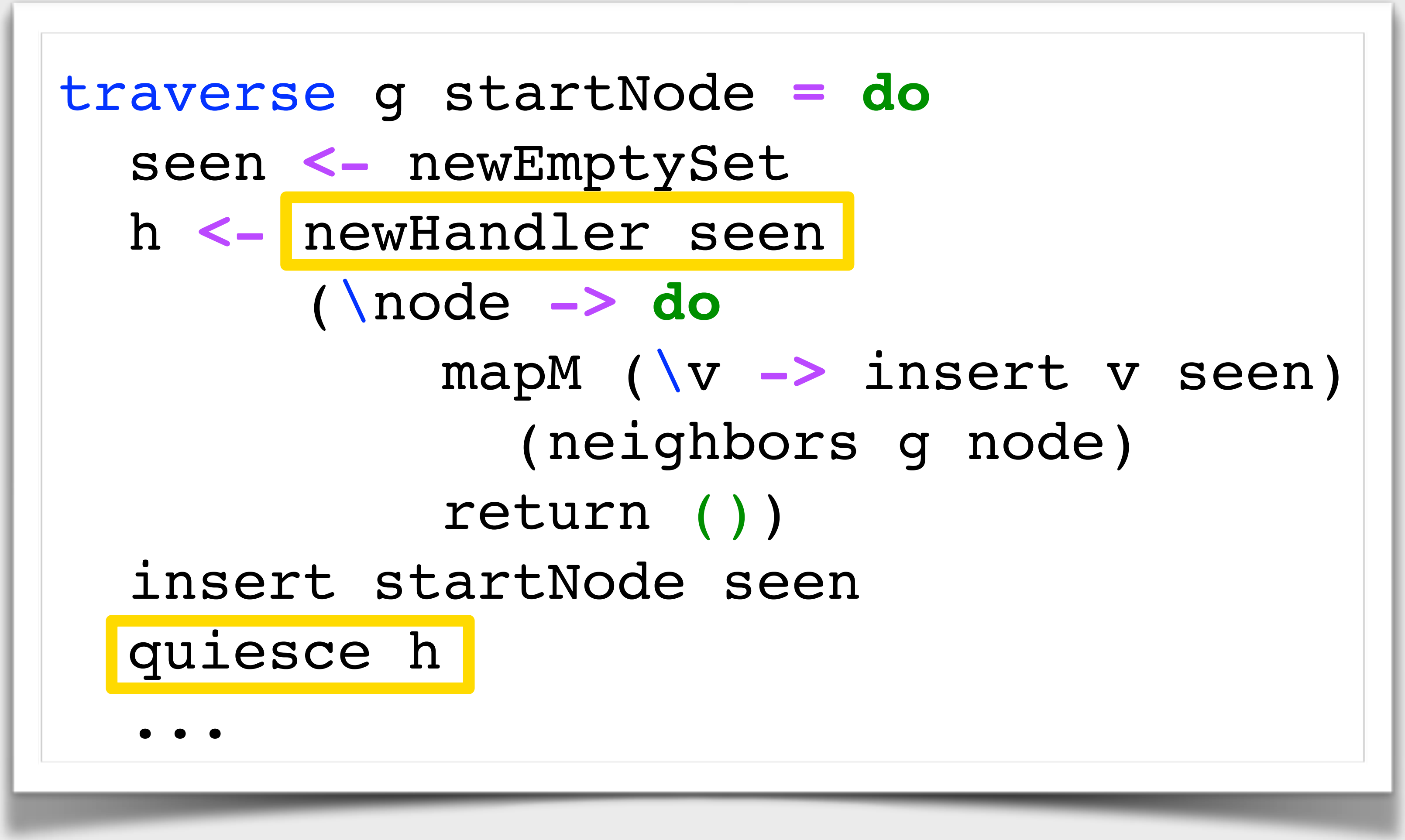
`freeze`: exact non-blocking read



```
traverse g startNode = do
  seen <- newEmptySet
  h <- newHandler seen
       (\node -> do
           mapM (\v -> insert v seen)
             (neighbors g node)
           return ())
  insert startNode seen
  quiesce h
  ...
```

`freeze`: exact non-blocking read

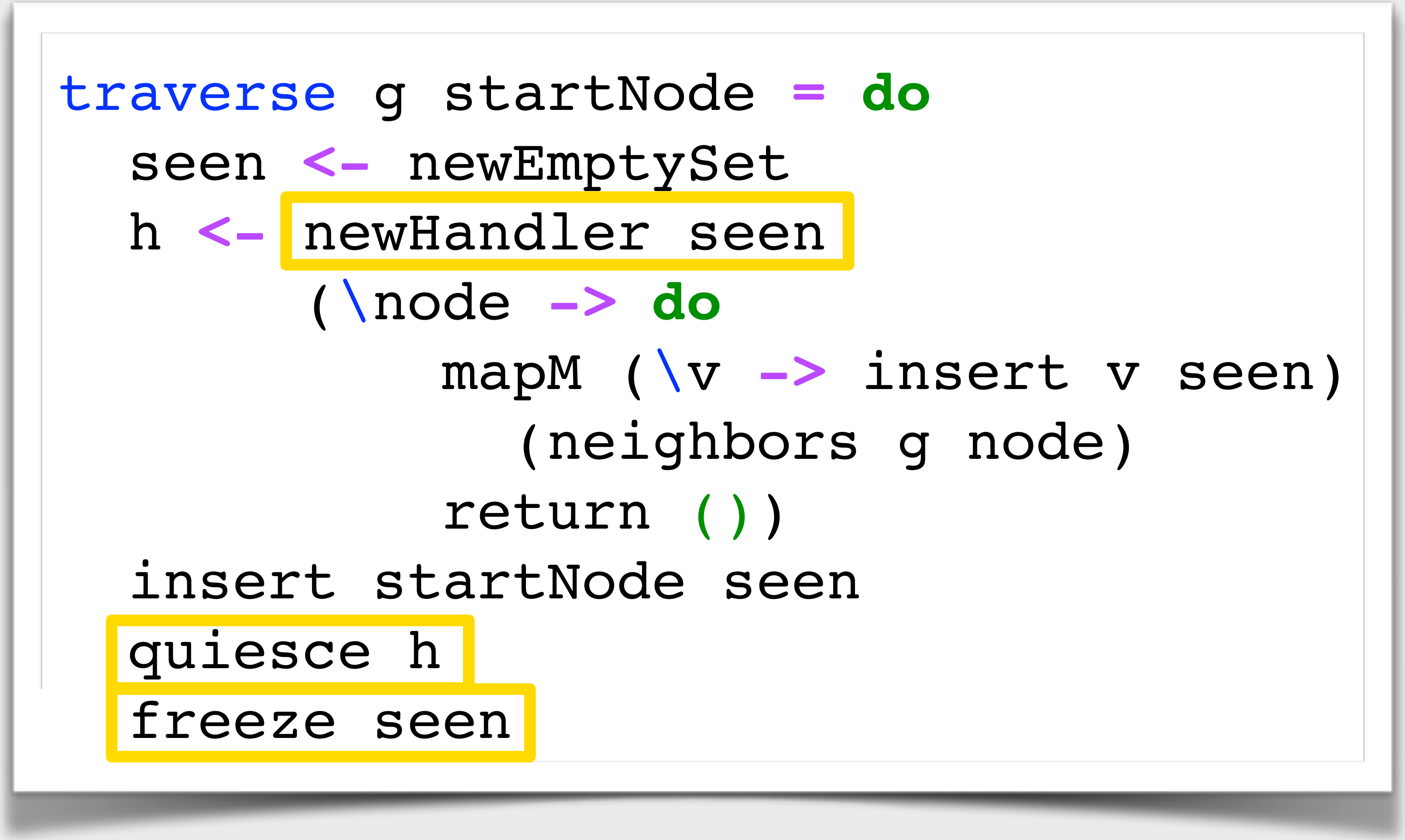
Attempts to write to a frozen LVar raise a write-after-`freeze` exception



```
traverse g startNode = do
  seen <- newEmptySet
  h <- newHandler seen
       (\node -> do
           mapM (\v -> insert v seen)
             (neighbors g node)
           return ())
  insert startNode seen
  quiesce h
  ...
```

`freeze`: exact non-blocking read

Attempts to write to a frozen LVar raise a write-after-`freeze` exception

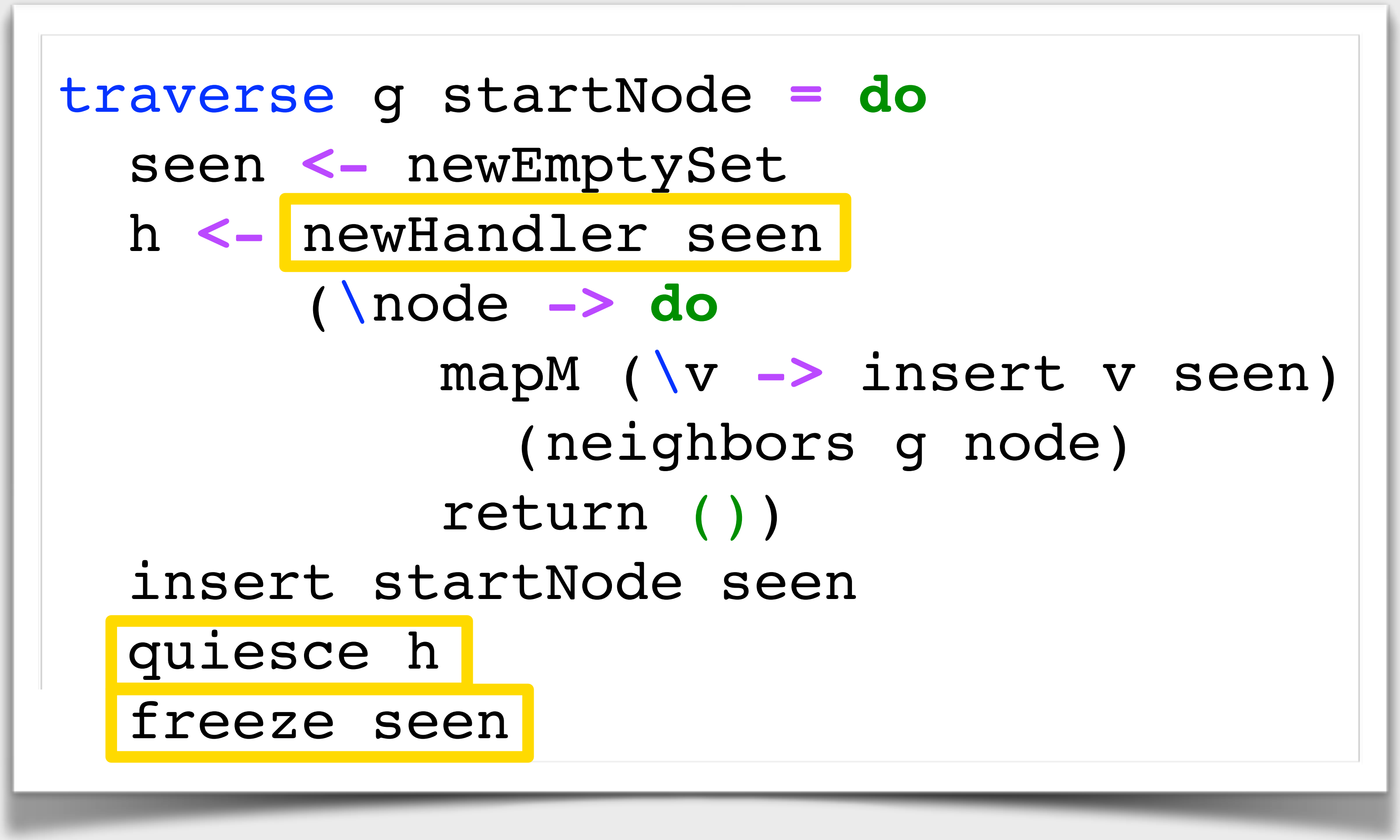


```
traverse g startNode = do
  seen <- newEmptySet
  h <- newHandler seen
       (\node -> do
           mapM (\v -> insert v seen)
             (neighbors g node)
           return ())
  insert startNode seen
  quiesce h
  freeze seen
```

`freeze`: exact non-blocking read

Attempts to write to a frozen LVar raise a write-after-`freeze` exception

Two possible outcomes: either the same final value or an exception

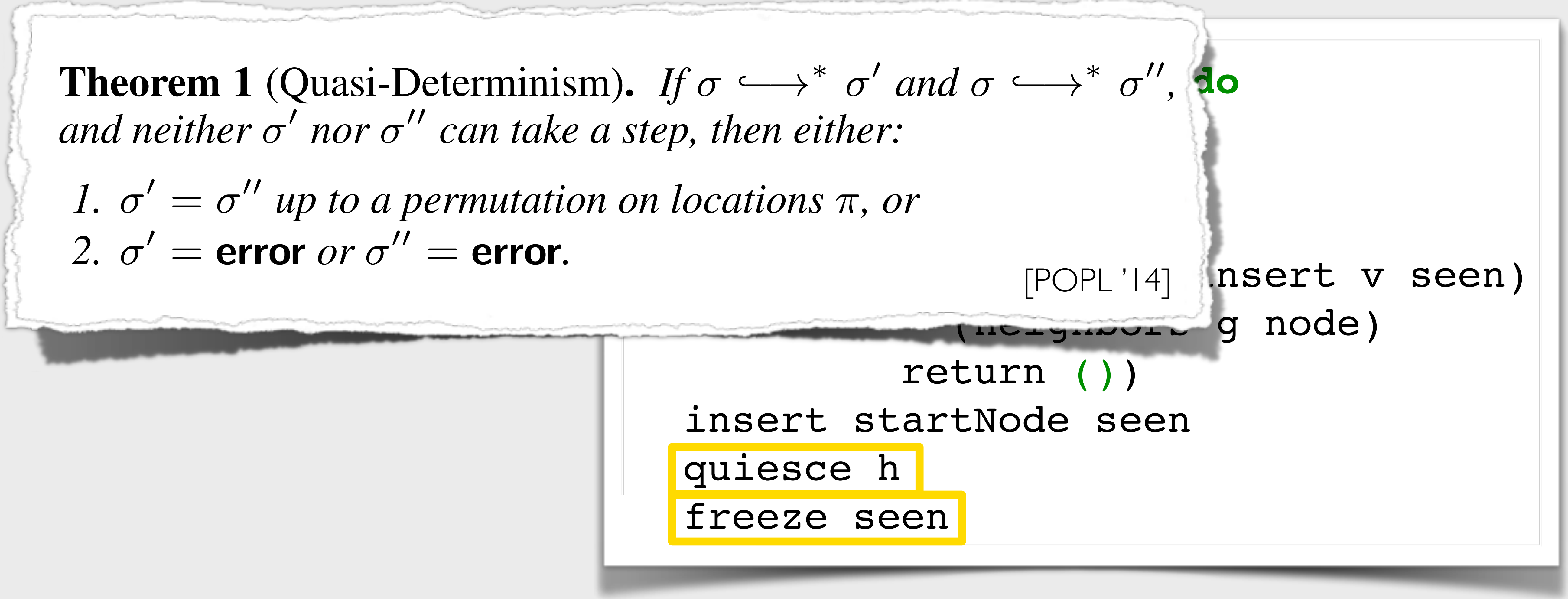


```
traverse g startNode = do
  seen <- newEmptySet
  h <- newHandler seen
          (\node -> do
              mapM (\v -> insert v seen)
                (neighbors g node)
              return ())
  insert startNode seen
  quiesce h
  freeze seen
```

`freeze`: exact non-blocking read

Attempts to write to a frozen LVar raise a write-after-`freeze` exception

Two possible outcomes: either the same final value or an exception



Theorem 1 (Quasi-Determinism). *If $\sigma \hookrightarrow^* \sigma'$ and $\sigma \hookrightarrow^* \sigma''$, and neither σ' nor σ'' can take a step, then either:*

1. *$\sigma' = \sigma''$ up to a permutation on locations π, or*
2. *$\sigma' =$ **error** or $\sigma'' =$ **error**.*

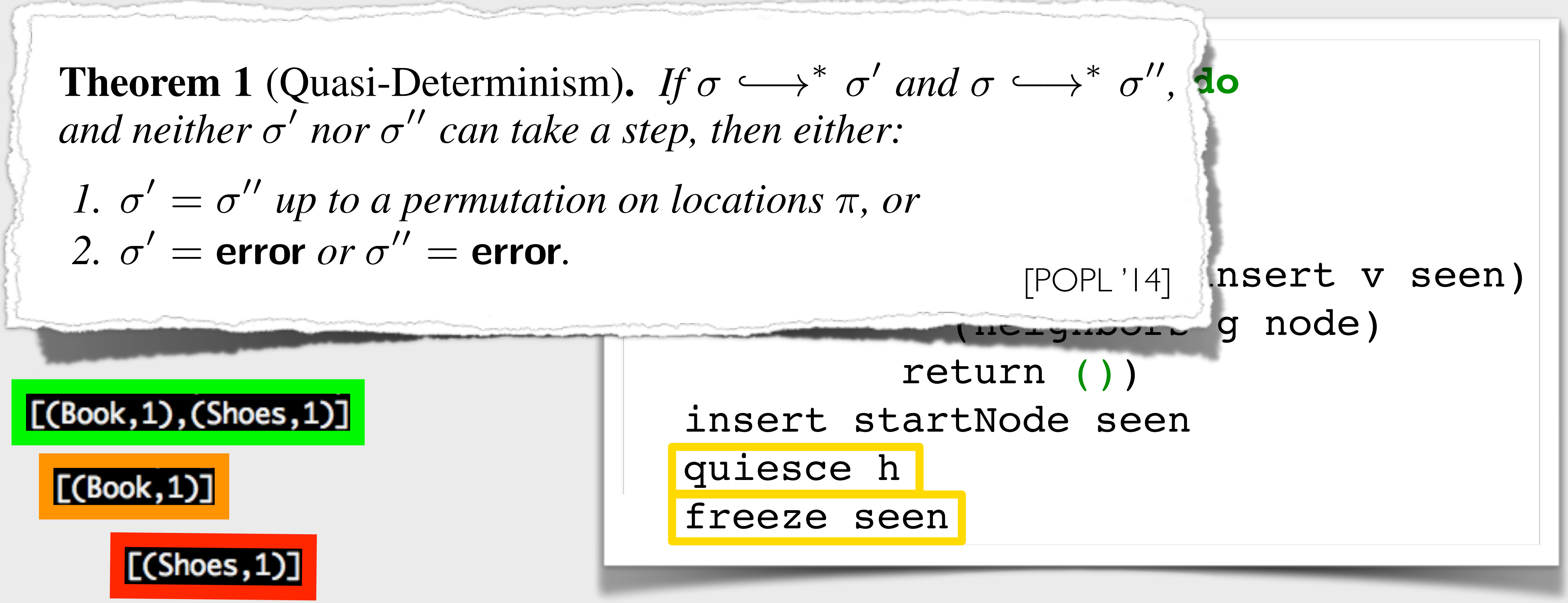
[POPL '14]

```
                     do
                     nsert v seen)
           (neighbors g node)
         return ())
insert startNode seen
quiesce h
freeze seen
```

`freeze`: exact non-blocking read

Attempts to write to a frozen LVar raise a write-after-`freeze` exception

Two possible outcomes: either the same final value or an exception

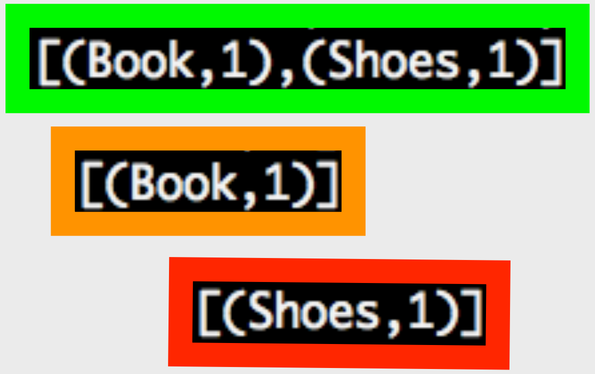


Theorem 1 (Quasi-Determinism). *If $\sigma \hookrightarrow^* \sigma'$ and $\sigma \hookrightarrow^* \sigma''$, and neither σ' nor σ'' can take a step, then either:*

1. *$\sigma' = \sigma''$ up to a permutation on locations π, or*
2. *$\sigma' =$ **error** or $\sigma'' =$ **error**.*

[POPL '14]

```
                                                 do



                                        nsert v seen)
                         (neighbors g node)
                         return ())
  insert startNode seen
  quiesce h
  freeze seen
```



```
[(Book,1),(Shoes,1)]
```

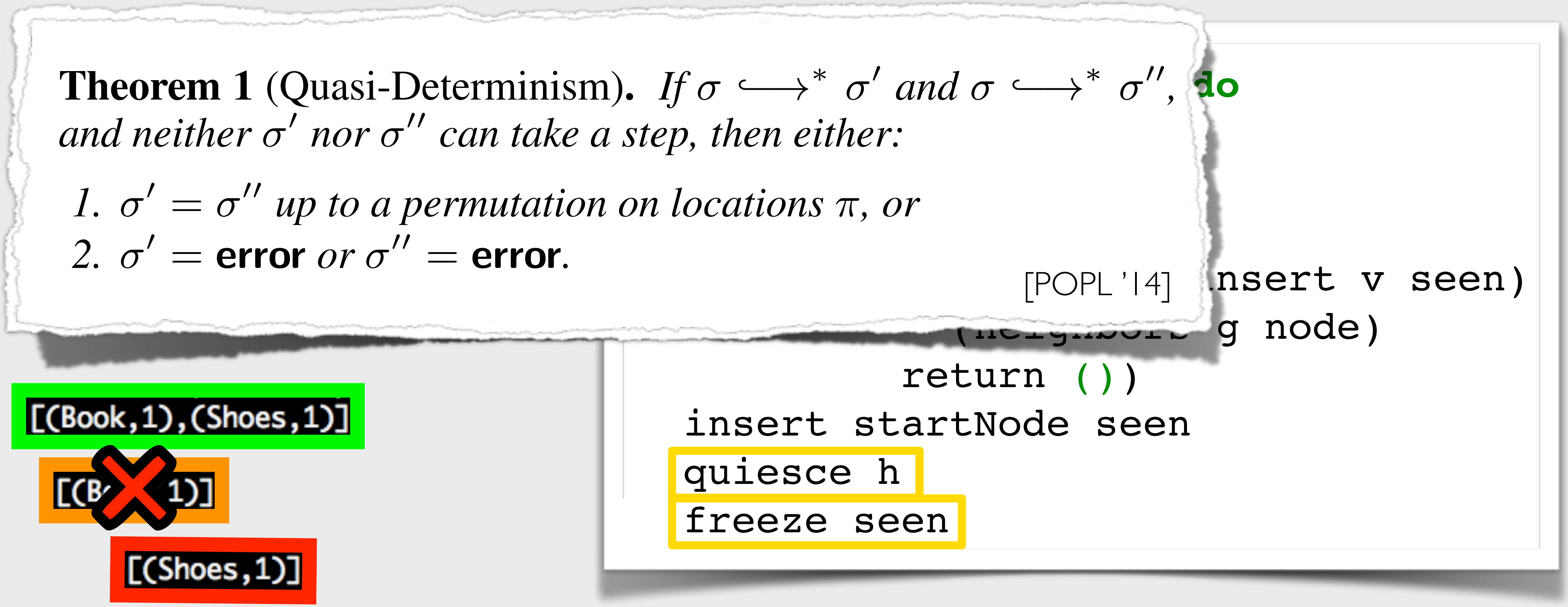
```
[(Book,1)]
```

```
[(Shoes,1)]
```

`freeze`: exact non-blocking read

Attempts to write to a frozen LVar raise a write-after-`freeze` exception

Two possible outcomes: either the same final value or an exception



Theorem 1 (Quasi-Determinism). *If $\sigma \hookrightarrow^* \sigma'$ and $\sigma \hookrightarrow^* \sigma''$, and neither σ' nor σ'' can take a step, then either:*

1. *$\sigma' = \sigma''$ up to a permutation on locations π, or*
2. *$\sigma' =$ **error** or $\sigma'' =$ **error**.*

[POPL '14]

```
                                    do




                                    nsert v seen)
                         (neighbors g node)
                  return ())
  insert startNode seen
  quiesce h
  freeze seen
```

```
[(Book,1),(Shoes,1)]
```

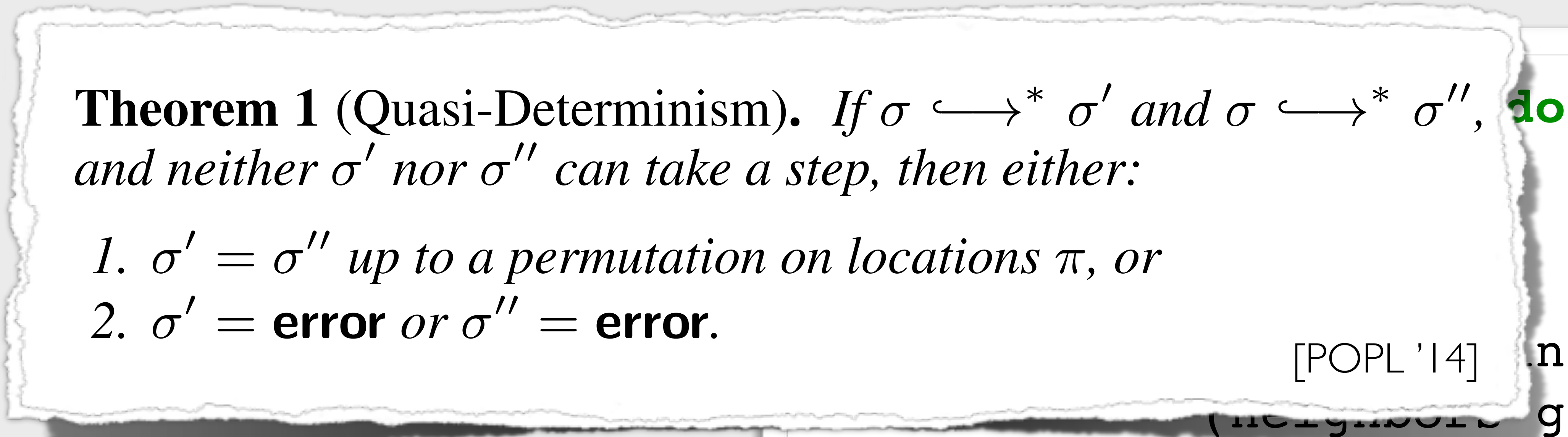
```
[(Book,1)]
```

```
[(Shoes,1)]
```

`freeze`: exact non-blocking read

Attempts to write to a frozen LVar raise a write-after-`freeze` exception

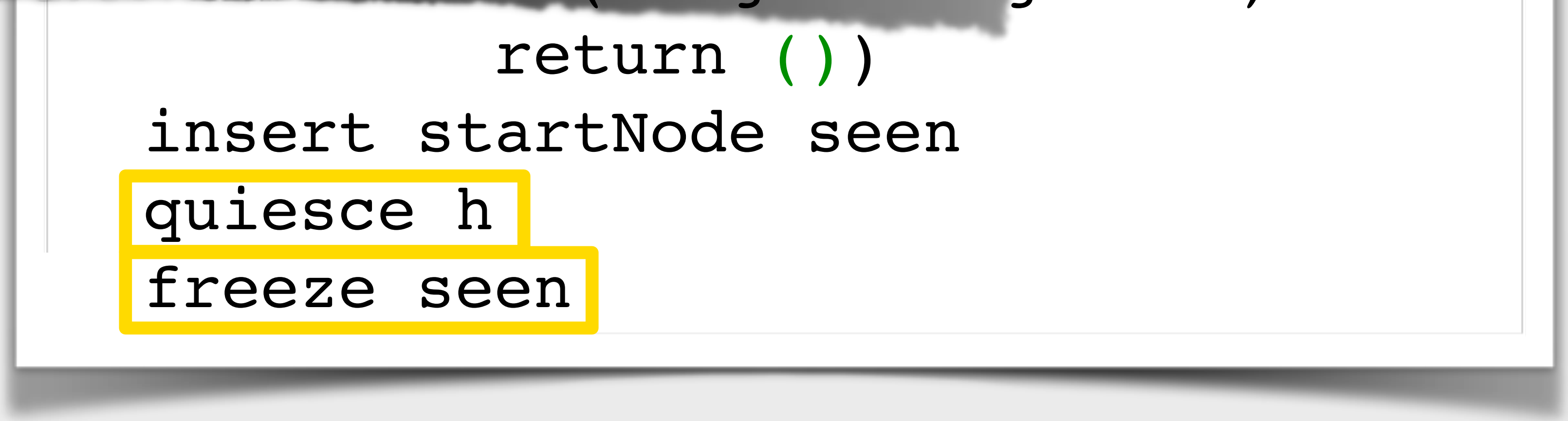
Two possible outcomes: either the same final value or an exception



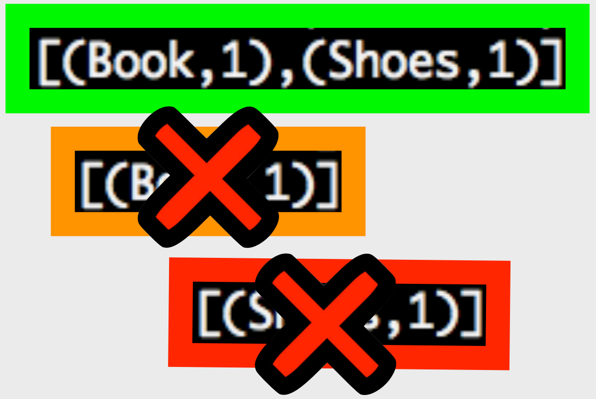
Theorem 1 (Quasi-Determinism). *If $\sigma \hookrightarrow^* \sigma'$ and $\sigma \hookrightarrow^* \sigma''$, and neither σ' nor σ'' can take a step, then either:*

1. *$\sigma' = \sigma''$ up to a permutation on locations π, or*
2. *$\sigma' =$ **error** or $\sigma'' =$ **error**.*

[POPL '14]



```
                                                         do




                                                         nsert v seen)
                     (neighbors g node)
                     return ())
  insert startNode seen
  quiesce h
  freeze seen
```

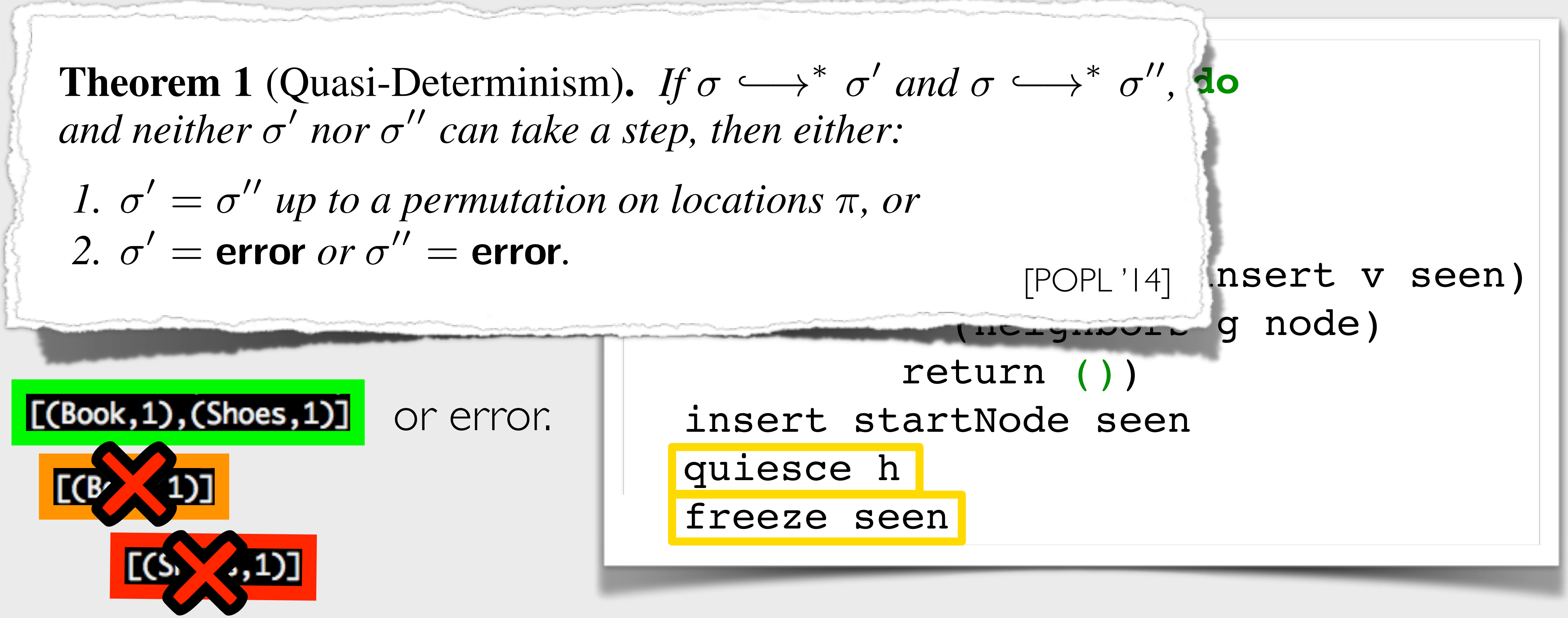


```
[(Book,1),(Shoes,1)]
 [(B     1)]
    [(S     ,1)]
```

`freeze`: exact non-blocking read

Attempts to write to a frozen LVar raise a write-after-`freeze` exception

Two possible outcomes: either the same final value or an exception



Theorem 1 (Quasi-Determinism). *If $\sigma \hookrightarrow^* \sigma'$ and $\sigma \hookrightarrow^* \sigma''$, and neither σ' nor σ'' can take a step, then either:*

1. *$\sigma' = \sigma''$ up to a permutation on locations π, or*
2. *$\sigma' =$ **error** or $\sigma'' =$ **error**.*

[POPL '14]

```
do
nsert v seen)
g node)
            return ())
insert startNode seen
quiesce h
freeze seen
```

[(Book,1),(Shoes,1)] or error.

[(B 1)]

[(S ,1)]

Determinism

$$\begin{tikzcd}
& \sigma \arrow[dl, "*"] \arrow[dr, "*"] & \\
\sigma' & = & \sigma''
\end{tikzcd}$$

[POPL '14]

Independence

$$\frac{\langle S;\, e\rangle \hookrightarrow \langle S';\, e'\rangle}{\langle S \sqcup_S S'';\, e\rangle \hookrightarrow \langle S' \sqcup_S S'';\, e'\rangle}$$

Determinism

$$\begin{tikzcd}
 & \sigma \arrow[dl, "*"] \arrow[dr, "*"] & \\
\sigma' & = & \sigma''
\end{tikzcd}$$

[POPL '14]

Independence

$$\frac{\langle S;\, e\rangle \hookrightarrow \langle S';\, e'\rangle}{\langle S \sqcup_S S'';\, e\rangle \hookrightarrow \langle S' \sqcup_S S'';\, e'\rangle}$$

Determinism

$$\begin{tikzcd} & \sigma \arrow[dl, "*"] \arrow[dr, "*"] & \\ \sigma' & = & \sigma'' \end{tikzcd}$$

[O'Hearn *et al.*, 2001]

Frame rule

$$\frac{\{p\}\ c\ \{q\}}{\{p * r\}\ c\ \{q * r\}}$$

[POPL '14]

Independence

$$\frac{\langle S;\, e\rangle \hookrightarrow \langle S';\, e'\rangle}{\langle S \sqcup_S S'';\, e\rangle \hookrightarrow \langle S' \sqcup_S S'';\, e'\rangle}$$

→

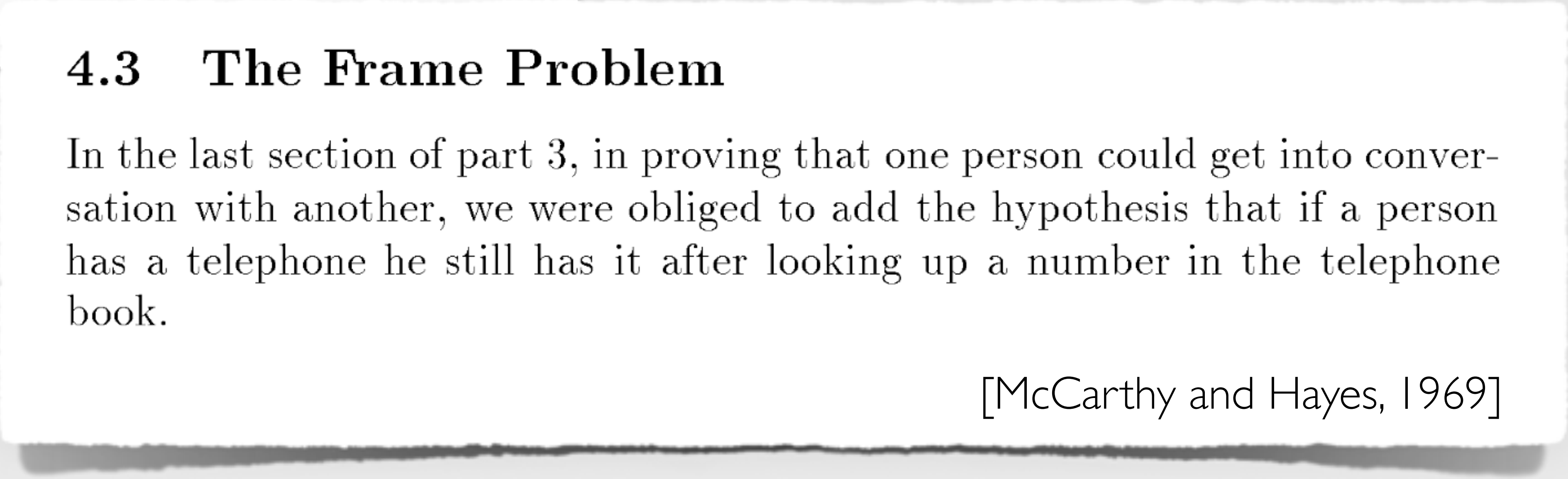
Determinism

$$\begin{tikzcd} & \sigma \arrow[dl, "*"] \arrow[dr, "*"] & \\ \sigma' & = & \sigma'' \end{tikzcd}$$

[O'Hearn *et al.*, 2001]

Frame rule

$$\frac{\{p\}\ c\ \{q\}}{\{p * r\}\ c\ \{q * r\}}$$



4.3 The Frame Problem

In the last section of part 3, in proving that one person could get into conversation with another, we were obliged to add the hypothesis that if a person has a telephone he still has it after looking up a number in the telephone book.

[McCarthy and Hayes, 1969]

[POPL '14]

Independence

$$\frac{\langle S;\ e\rangle \hookrightarrow \langle S';\ e'\rangle}{\langle S \sqcup_S S'';\ e\rangle \hookrightarrow \langle S' \sqcup_S S'';\ e'\rangle}$$

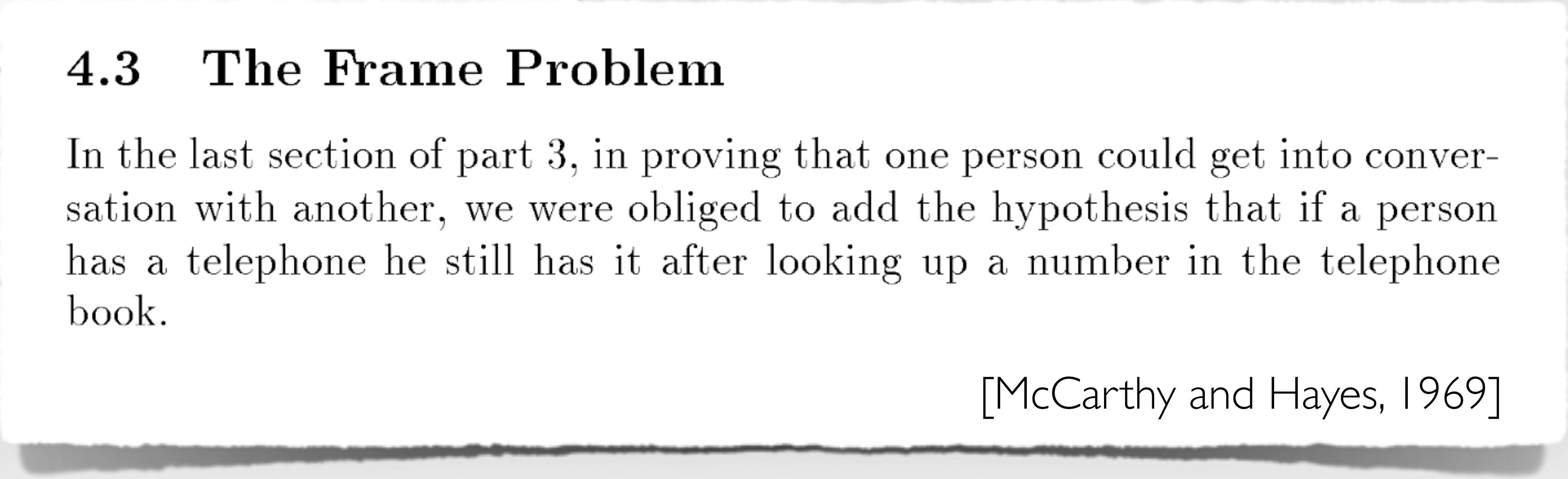
Determinism

$$\begin{tikzcd} & \sigma \arrow[dl, "*"'] \arrow[dr, "*"] & \\ \sigma' & = & \sigma'' \end{tikzcd}$$

[O'Hearn *et al.*, 2001]

Frame rule

$$\frac{\{p\}\ c\ \{q\}}{\{p * r\}\ c\ \{q * r\}}$$

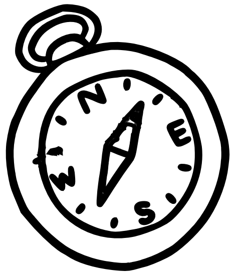


4.3 The Frame Problem

In the last section of part 3, in proving that one person could get into conversation with another, we were obliged to add the hypothesis that if a person has a telephone he still has it after looking up a number in the telephone book.

[McCarthy and Hayes, 1969]

[POPL '14]



Independence

$$\frac{\langle S;\ e\rangle \hookrightarrow \langle S';\ e'\rangle}{\langle S \sqcup_S S'';\ e\rangle \hookrightarrow \langle S' \sqcup_S S'';\ e'\rangle}$$

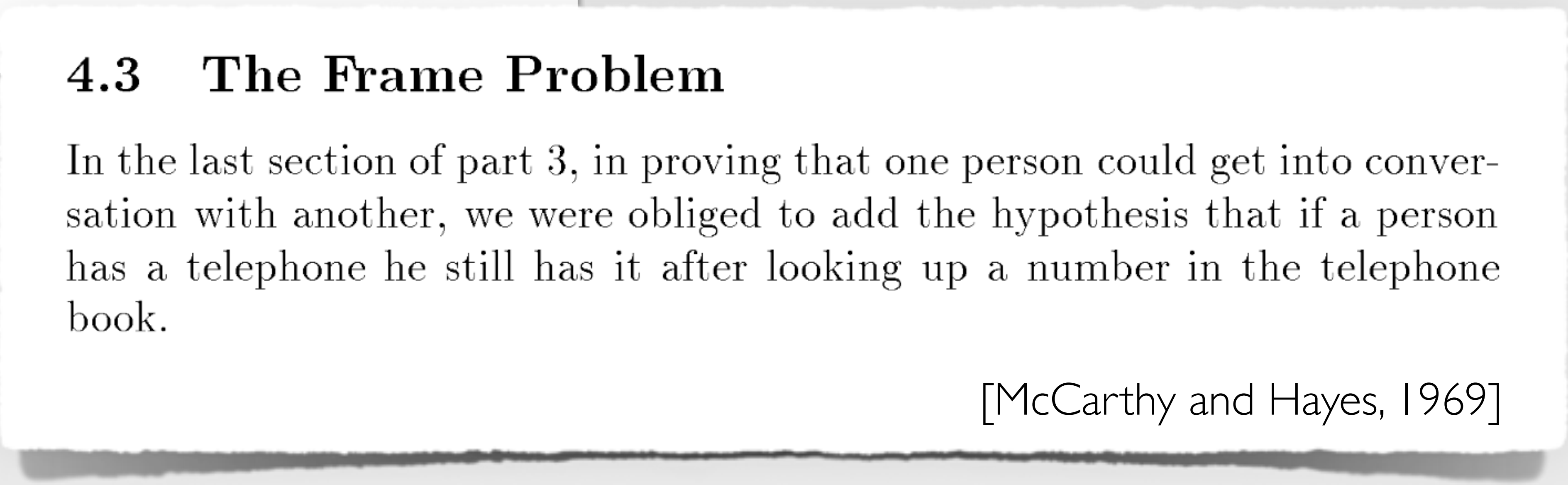
Determinism

$$\begin{tikzcd}
 & \sigma \arrow[dl, "*"'] \arrow[dr, "*"] & \\
\sigma' & = & \sigma''
\end{tikzcd}$$

[O'Hearn *et al.*, 2001]

Frame rule

$$\frac{\{p\}\ c\ \{q\}}{\{p * r\}\ c\ \{q * r\}}$$



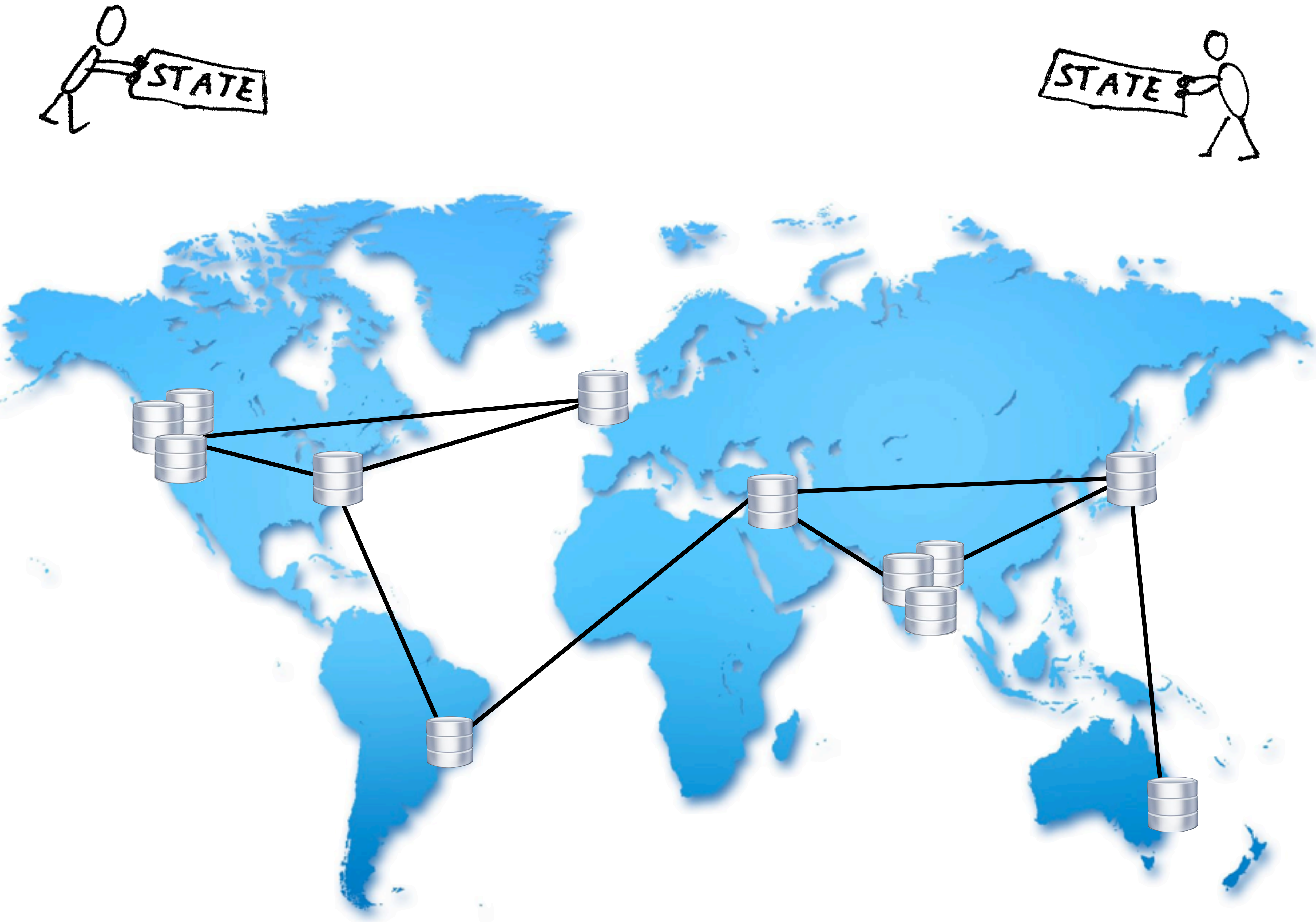
4.3 The Frame Problem

In the last section of part 3, in proving that one person could get into conversation with another, we were obliged to add the hypothesis that if a person has a telephone he still has it after looking up a number in the telephone book.

[McCarthy and Hayes, 1969]

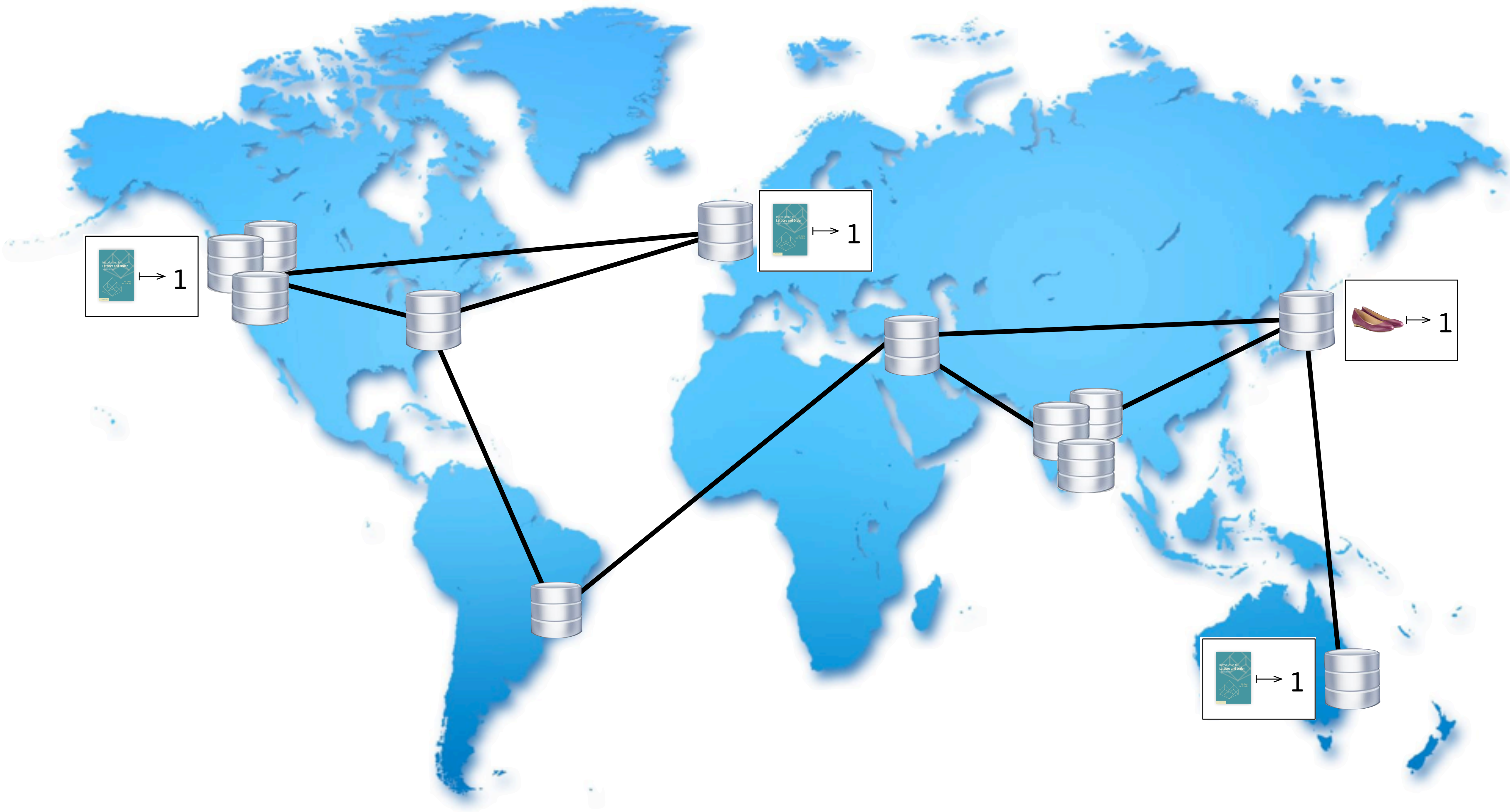
STATE

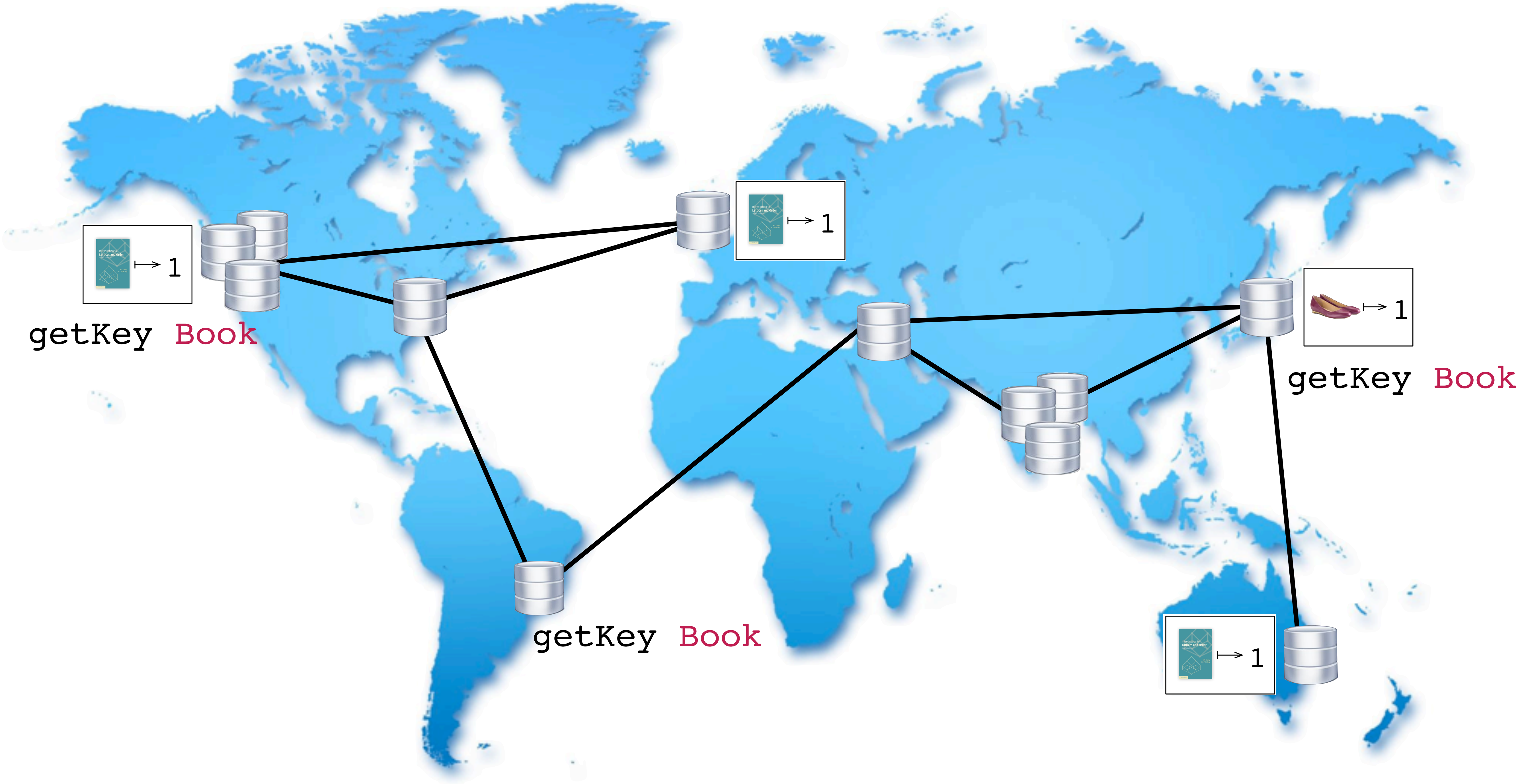
STATE


STATE
STATE

STATE

STATE

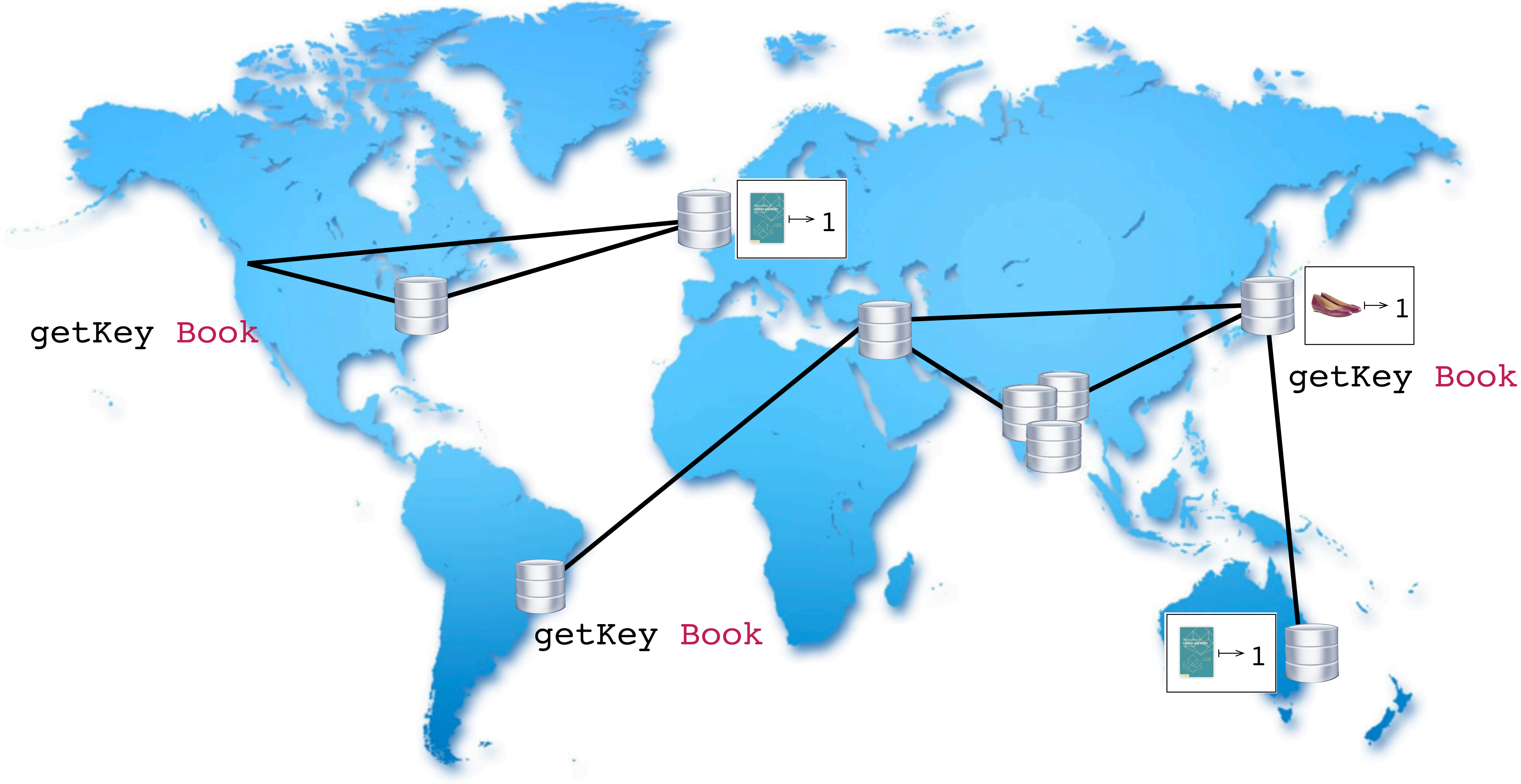

↦ 1
↦ 1
↦ 1
↦ 1



getKey Book

getKey Book

getKey Book

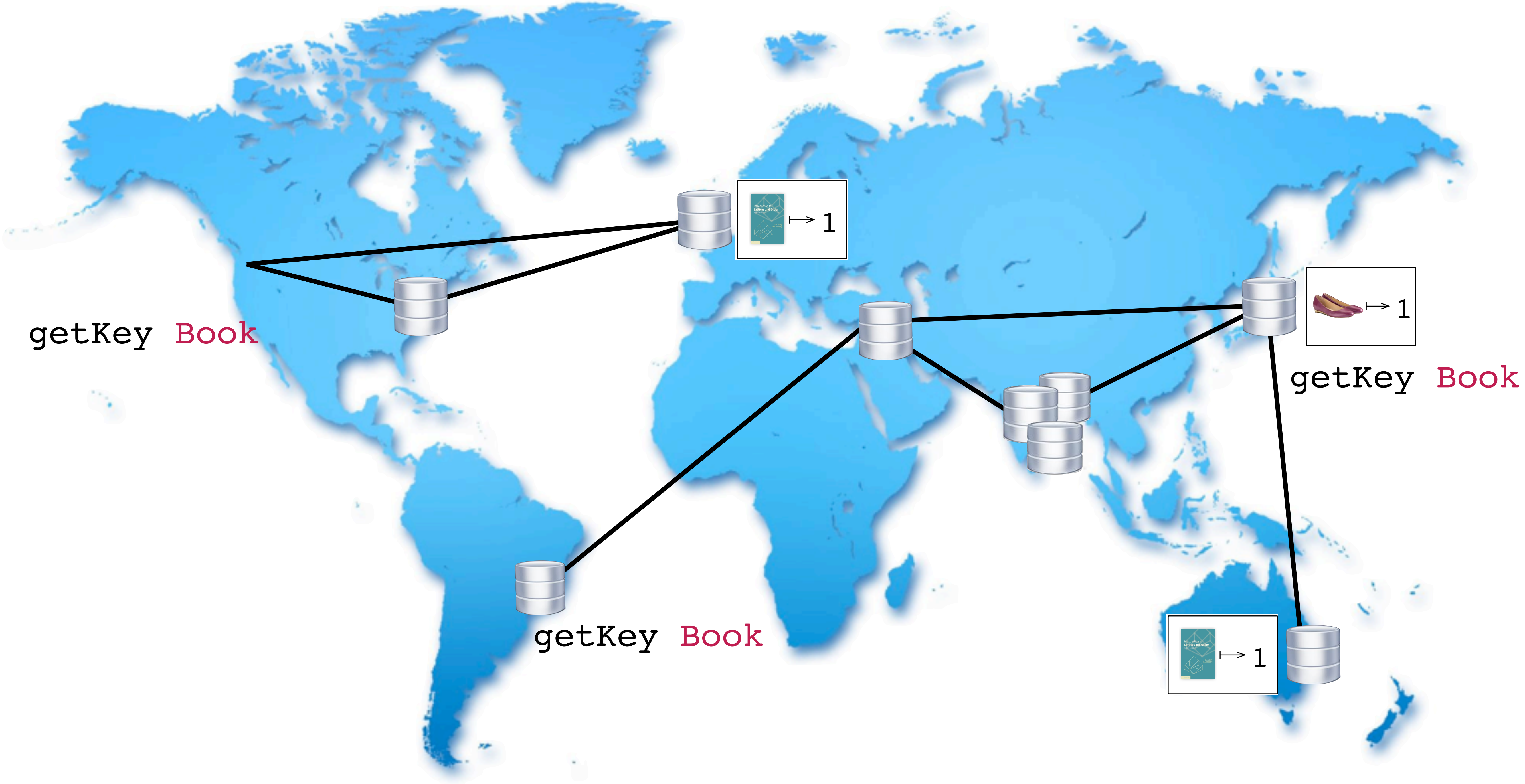


getKey Book

getKey Book

getKey Book

← *eventual* consistency →



getKey Book

↦ 1

↦ 1

getKey Book

getKey Book

↦ 1

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati,
Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall
and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management; D.4.5 [**Operating Systems**]: Reliability; D.4.2 [**Operating Systems**]: Performance;

General Terms

Algorithms, Management, Measurement, Performance, Design, Reliability.

1. INTRODUCTION

Amazon runs a world-wide e-commerce platform that serves tens of millions customers at peak times using tens of thousands of servers located in many data centers around the world. There are strict operational requirements on Amazon's platform in terms of performance, reliability and efficiency, and to support continuous growth the platform needs to be highly scalable. Reliability is one of the most important requirements because even the slightest outage has significant financial consequences and impacts customer trust. In addition, to support continuous growth, the platform needs to be highly scalable.

SOSP'07, October 14–17, 2007, Stevenson, Washington, USA.
Copyright 2007 ACM 978-1-59593-591-5/07/0010...$5.00.

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

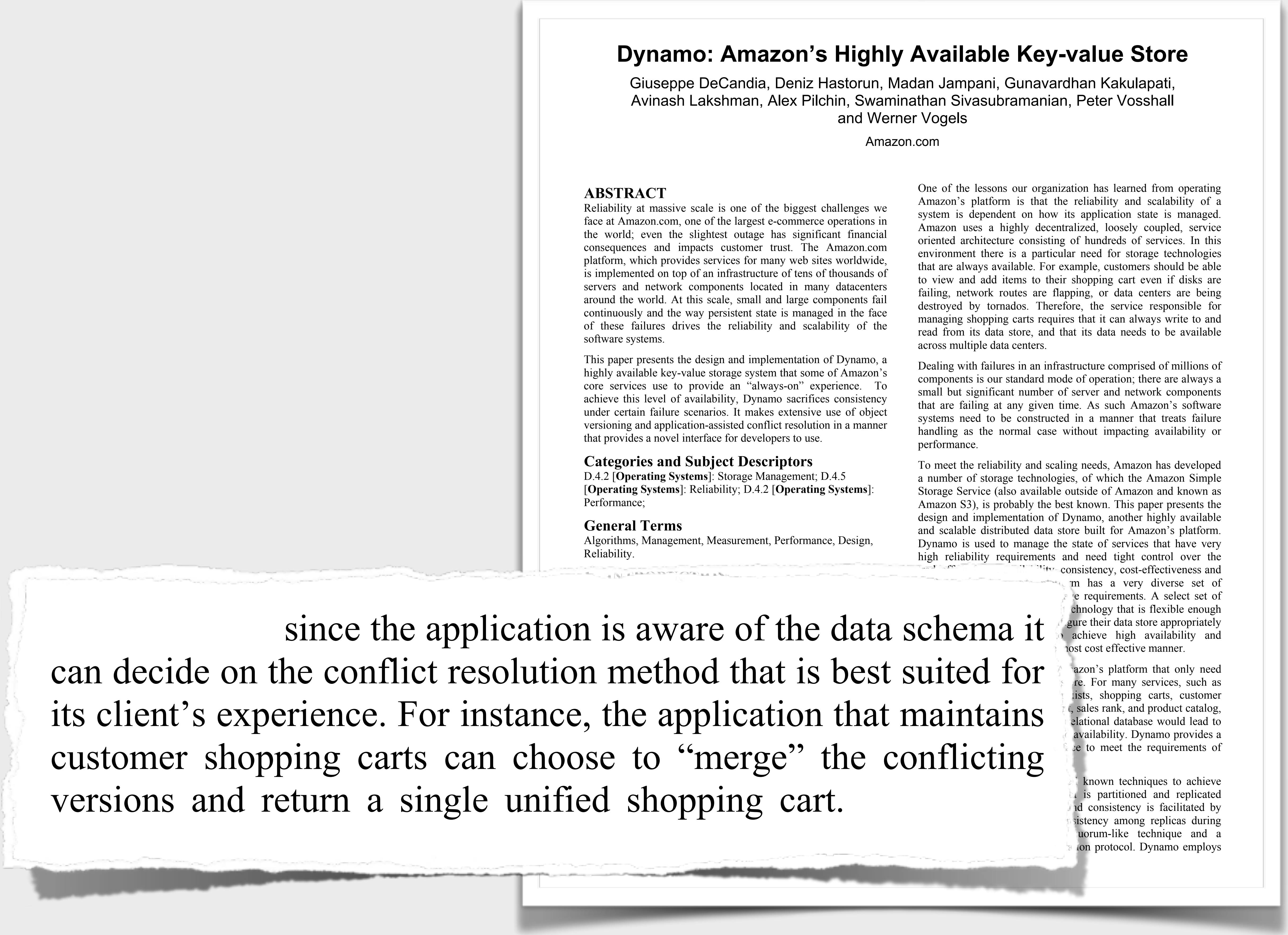
Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon's software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

To meet the reliability and scaling needs, Amazon has developed a number of storage technologies, of which the Amazon Simple Storage Service (also available outside of Amazon and known as Amazon S3), is probably the best known. This paper presents the design and implementation of Dynamo, another highly available and scalable distributed data store built for Amazon's platform. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performance. Amazon's platform has a very diverse set of applications with different storage requirements. A select set of applications requires a storage technology that is flexible enough to let application designers configure their data store appropriately based on these tradeoffs to achieve high availability and guaranteed performance in the most cost effective manner.

There are many services on Amazon's platform that only need primary-key access to a data store. For many services, such as those that provide best seller lists, shopping carts, customer preferences, session management, sales rank, and product catalog, the common pattern of using a relational database would lead to inefficiencies and limit scale and availability. Dynamo provides a simple primary-key only interface to meet the requirements of these applications.

Dynamo uses a synthesis of well known techniques to achieve scalability and availability: Data is partitioned and replicated using consistent hashing [10], and consistency is facilitated by object versioning [12]. The consistency among replicas during updates is maintained by a quorum-like technique and a decentralized replica synchronization protocol. Dynamo employs

[DeCandia *et al*., SOSP '07]



Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati,
Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall
and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management; D.4.5 [**Operating Systems**]: Reliability; D.4.2 [**Operating Systems**]: Performance;

General Terms

Algorithms, Management, Measurement, Performance, Design, Reliability.

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon's software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

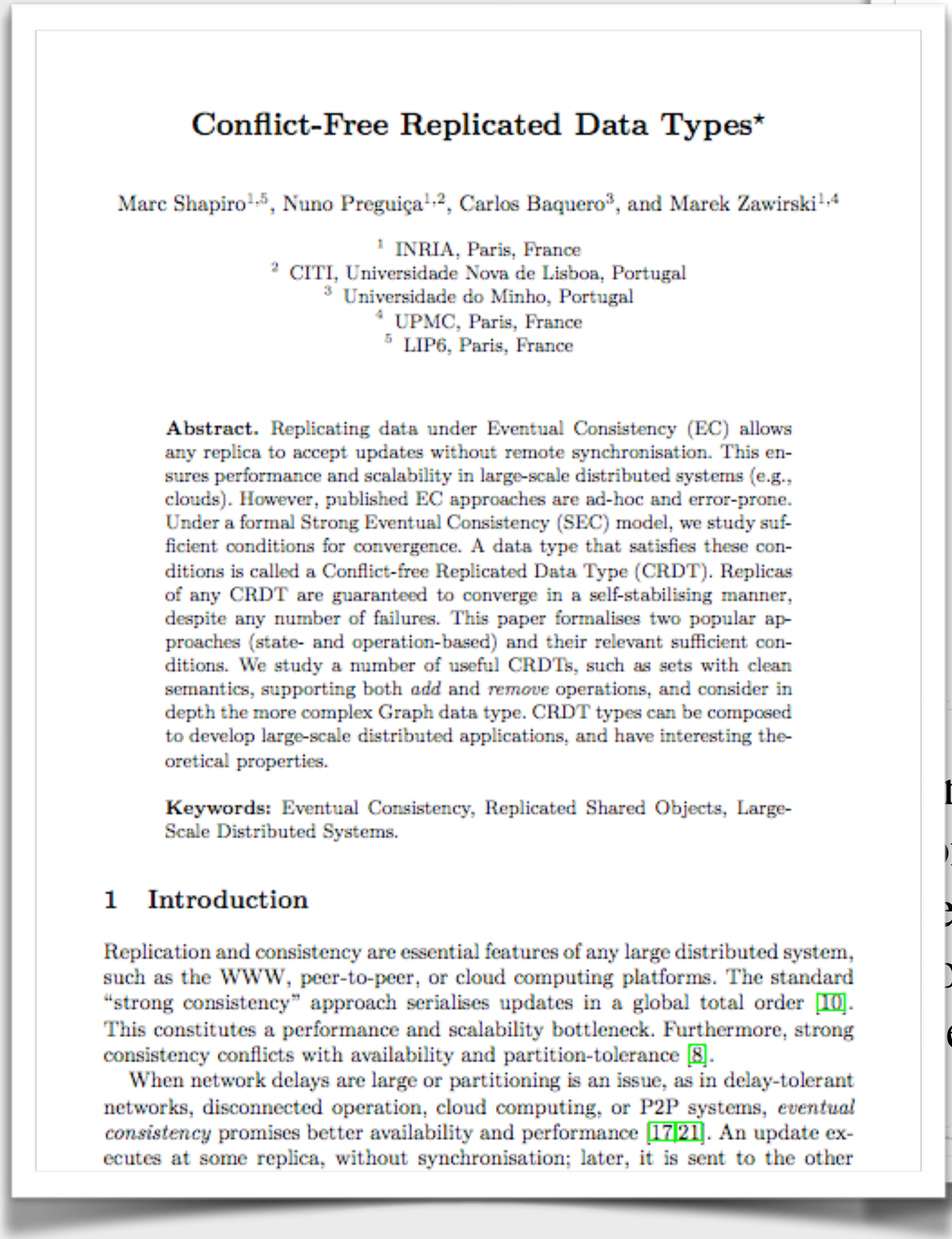
To meet the reliability and scaling needs, Amazon has developed a number of storage technologies, of which the Amazon Simple Storage Service (also available outside of Amazon and known as Amazon S3), is probably the best known. This paper presents the design and implementation of Dynamo, another highly available and scalable distributed data store built for Amazon's platform. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the
consistency, cost-effectiveness and
m has a very diverse set of
e requirements. A select set of
chnology that is flexible enough
gure their data store appropriately
achieve high availability and
ost cost effective manner.

azon's platform that only need
re. For many services, such as
ists, shopping carts, customer
, sales rank, and product catalog,
elational database would lead to
availability. Dynamo provides a
e to meet the requirements of

known techniques to achieve
is partitioned and replicated
d consistency is facilitated by
sistency among replicas during
uorum-like technique and a
ion protocol. Dynamo employs

since the application is aware of the data schema it can decide on the conflict resolution method that is best suited for its client's experience. For instance, the application that maintains customer shopping carts can choose to "merge" the conflicting versions and return a single unified shopping cart.

[DeCandia *et al*., SOSP '07]



Conflict-Free Replicated Data Types*

Marc Shapiro[1,5], Nuno Preguiça[1,2], Carlos Baquero[3], and Marek Zawirski[1,4]

[1] INRIA, Paris, France
[2] CITI, Universidade Nova de Lisboa, Portugal
[3] Universidade do Minho, Portugal
[4] UPMC, Paris, France
[5] LIP6, Paris, France

Abstract. Replicating data under Eventual Consistency (EC) allows any replica to accept updates without remote synchronisation. This ensures performance and scalability in large-scale distributed systems (e.g., clouds). However, published EC approaches are ad-hoc and error-prone. Under a formal Strong Eventual Consistency (SEC) model, we study sufficient conditions for convergence. A data type that satisfies these conditions is called a Conflict-free Replicated Data Type (CRDT). Replicas of any CRDT are guaranteed to converge in a self-stabilising manner, despite any number of failures. This paper formalises two popular approaches (state- and operation-based) and their relevant sufficient conditions. We study a number of useful CRDTs, such as sets with clean semantics, supporting both *add* and *remove* operations, and consider in depth the more complex Graph data type. CRDT types can be composed to develop large-scale distributed applications, and have interesting theoretical properties.

Keywords: Eventual Consistency, Replicated Shared Objects, Large-Scale Distributed Systems.

1 Introduction

Replication and consistency are essential features of any large distributed system, such as the WWW, peer-to-peer, or cloud computing platforms. The standard "strong consistency" approach serialises updates in a global total order [10]. This constitutes a performance and scalability bottleneck. Furthermore, strong consistency conflicts with availability and partition-tolerance [8].

When network delays are large or partitioning is an issue, as in delay-tolerant networks, disconnected operation, cloud computing, or P2P systems, *eventual consistency* promises better availability and performance [17,21]. An update executes at some replica, without synchronisation; later, it is sent to the other

[Shapiro *et al.*, SSS ’11]

Dynamo: Amazon’s Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati,
Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall
and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon’s core services use to provide an “always-on” experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management; D.4.5 [**Operating Systems**]: Reliability; D.4.2 [**Operating Systems**]: Performance;

General Terms

Algorithms, Management, Measurement, Performance, Design, Reliability.

One of the lessons our organization has learned from operating Amazon’s platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon’s software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

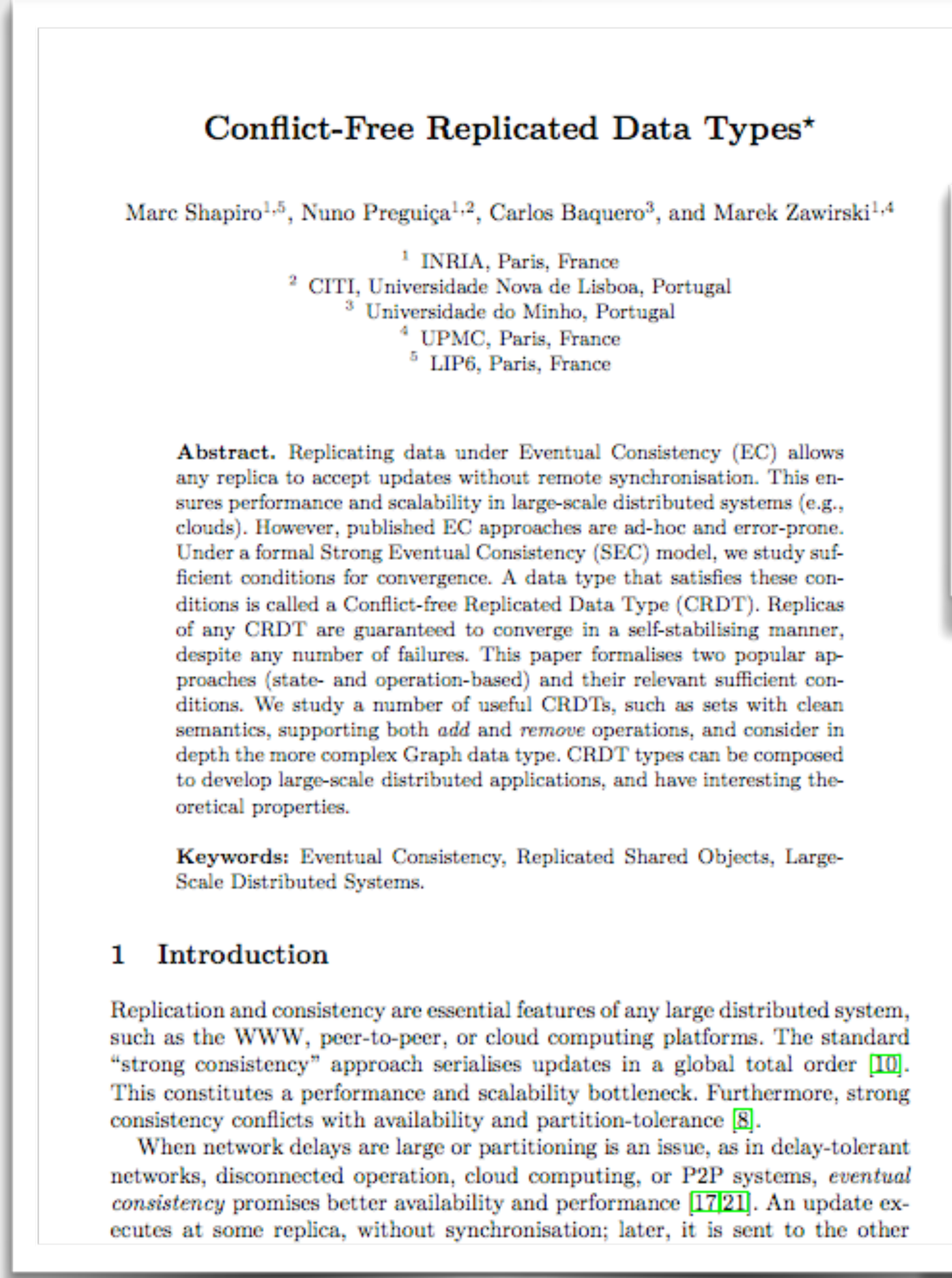
To meet the reliability and scaling needs, Amazon has developed a number of storage technologies, of which the Amazon Simple Storage Service (also available outside of Amazon and known as Amazon S3), is probably the best known. This paper presents the design and implementation of Dynamo, another highly available and scalable distributed data store built for Amazon’s platform. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the
consistency, cost-effectiveness and
m has a very diverse set of
e requirements. A select set of
chnology that is flexible enough
gure their data store appropriately
achieve high availability and
ost cost effective manner.

azon’s platform that only need
re. For many services, such as
ists, shopping carts, customer
, sales rank, and product catalog,
elational database would lead to
availability. Dynamo provides a
e to meet the requirements of

known techniques to achieve
is partitioned and replicated
nd consistency is facilitated by
sistency among replicas during
uorum-like technique and a
ion protocol. Dynamo employs

n is aware of the data schema it
n method that is best suited for
, the application that maintains
se to “merge” the conflicting
ed shopping cart.

[DeCandia *et al.*, SOSP ’07]



Conflict-Free Replicated Data Types*

Marc Shapiro[1,5], Nuno Preguiça[1,2], Carlos Baquero[3], and Marek Zawirski[1,4]

[1] INRIA, Paris, France
[2] CITI, Universidade Nova de Lisboa, Portugal
[3] Universidade do Minho, Portugal
[4] UPMC, Paris, France
[5] LIP6, Paris, France

Abstract. Replicating data under Eventual Consistency (EC) allows any replica to accept updates without remote synchronisation. This ensures performance and scalability in large-scale distributed systems (e.g., clouds). However, published EC approaches are ad-hoc and error-prone. Under a formal Strong Eventual Consistency (SEC) model, we study sufficient conditions for convergence. A data type that satisfies these conditions is called a Conflict-free Replicated Data Type (CRDT). Replicas of any CRDT are guaranteed to converge in a self-stabilising manner, despite any number of failures. This paper formalises two popular approaches (state- and operation-based) and their relevant sufficient conditions. We study a number of useful CRDTs, such as sets with clean semantics, supporting both *add* and *remove* operations, and consider in depth the more complex Graph data type. CRDT types can be composed to develop large-scale distributed applications, and have interesting theoretical properties.

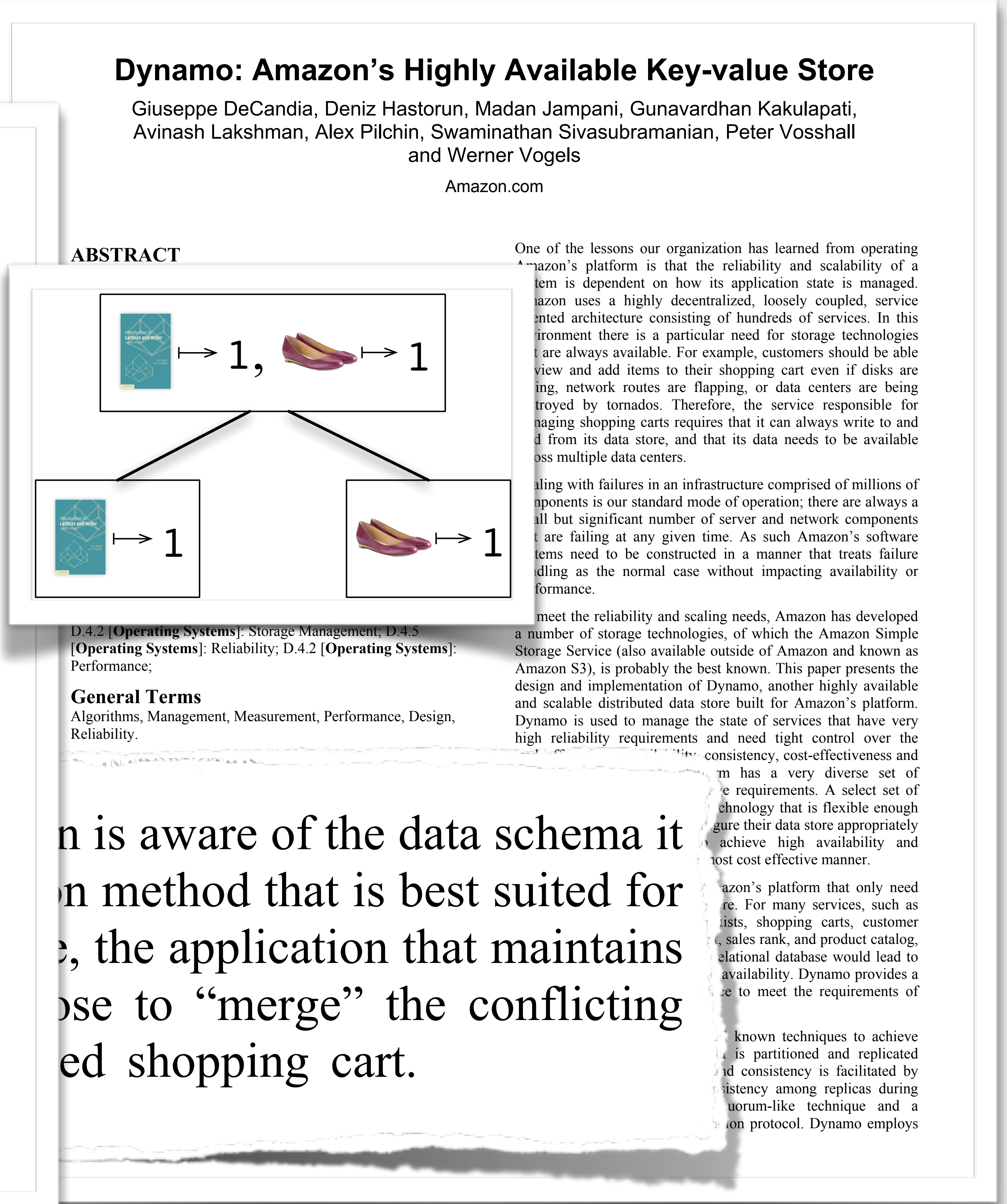
Keywords: Eventual Consistency, Replicated Shared Objects, Large-Scale Distributed Systems.

1 Introduction

Replication and consistency are essential features of any large distributed system, such as the WWW, peer-to-peer, or cloud computing platforms. The standard "strong consistency" approach serialises updates in a global total order [10]. This constitutes a performance and scalability bottleneck. Furthermore, strong consistency conflicts with availability and partition-tolerance [8].

When network delays are large or partitioning is an issue, as in delay-tolerant networks, disconnected operation, cloud computing, or P2P systems, *eventual consistency* promises better availability and performance [17,21]. An update executes at some replica, without synchronisation; later, it is sent to the other

[Shapiro *et al.*, SSS '11]



Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

ABSTRACT

D.4.2 [**Operating Systems**]: Storage Management; D.4.5 [**Operating Systems**]: Reliability; D.4.2 [**Operating Systems**]: Performance;

General Terms

Algorithms, Management, Measurement, Performance, Design, Reliability.

n is aware of the data schema it
n method that is best suited for
, the application that maintains
se to "merge" the conflicting
ed shopping cart.

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a
tem is dependent on how its application state is managed.
nazon uses a highly decentralized, loosely coupled, service
ented architecture consisting of hundreds of services. In this
ironment there is a particular need for storage technologies
t are always available. For example, customers should be able
view and add items to their shopping cart even if disks are
ing, network routes are flapping, or data centers are being
troyed by tornados. Therefore, the service responsible for
naging shopping carts requires that it can always write to and
d from its data store, and that its data needs to be available
oss multiple data centers.

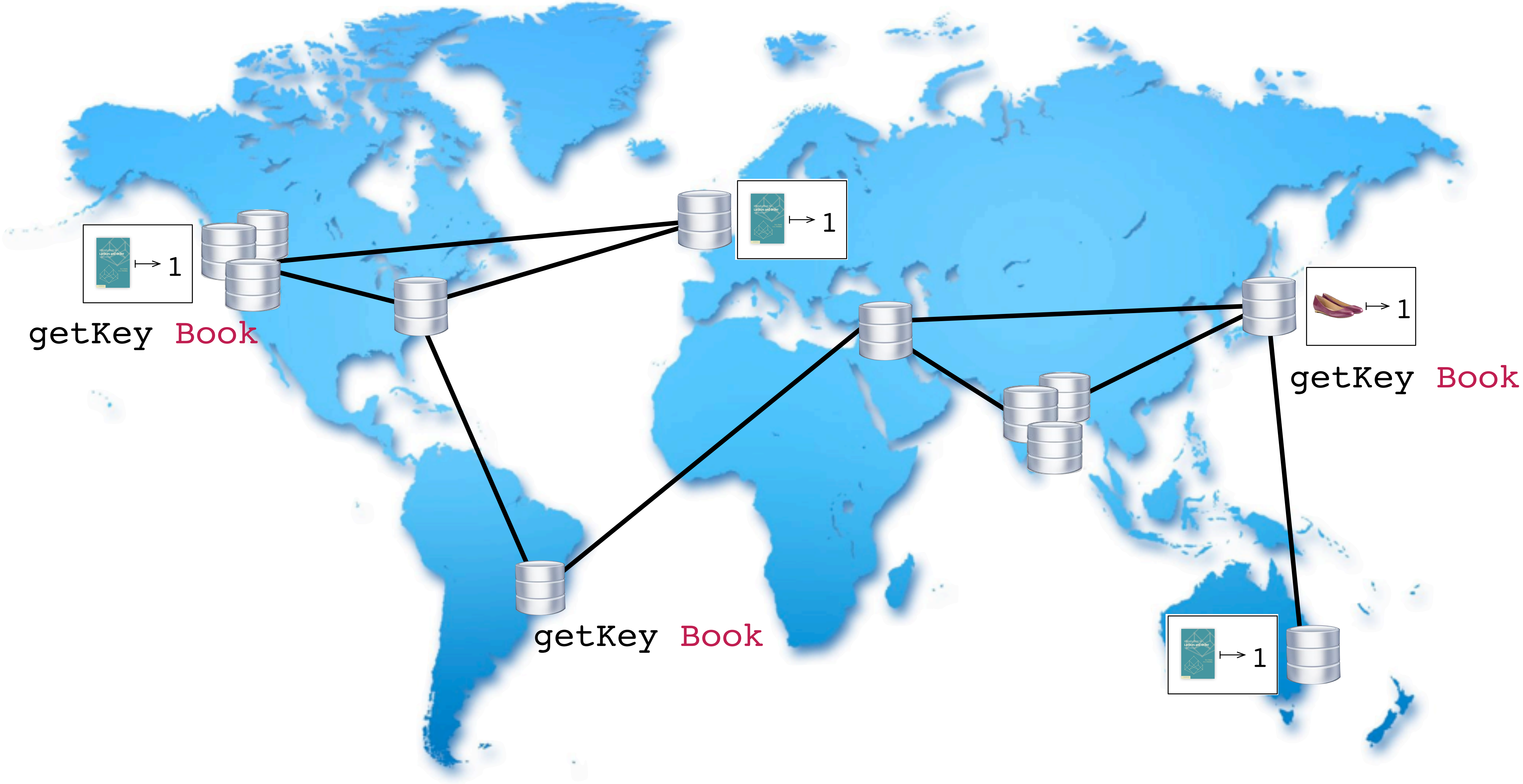
aling with failures in an infrastructure comprised of millions of
nponents is our standard mode of operation; there are always a
all but significant number of server and network components
t are failing at any given time. As such Amazon's software
tems need to be constructed in a manner that treats failure
ndling as the normal case without impacting availability or
formance.

meet the reliability and scaling needs, Amazon has developed a number of storage technologies, of which the Amazon Simple Storage Service (also available outside of Amazon and known as Amazon S3), is probably the best known. This paper presents the design and implementation of Dynamo, another highly available and scalable distributed data store built for Amazon's platform. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the
consistency, cost-effectiveness and
m has a very diverse set of
e requirements. A select set of
chnology that is flexible enough
gure their data store appropriately
achieve high availability and
nost cost effective manner.

azon's platform that only need
re. For many services, such as
lists, shopping carts, customer
, sales rank, and product catalog,
elational database would lead to
availability. Dynamo provides a
e to meet the requirements of

known techniques to achieve
is partitioned and replicated
nd consistency is facilitated by
sistency among replicas during
uorum-like technique and a
on protocol. Dynamo employs

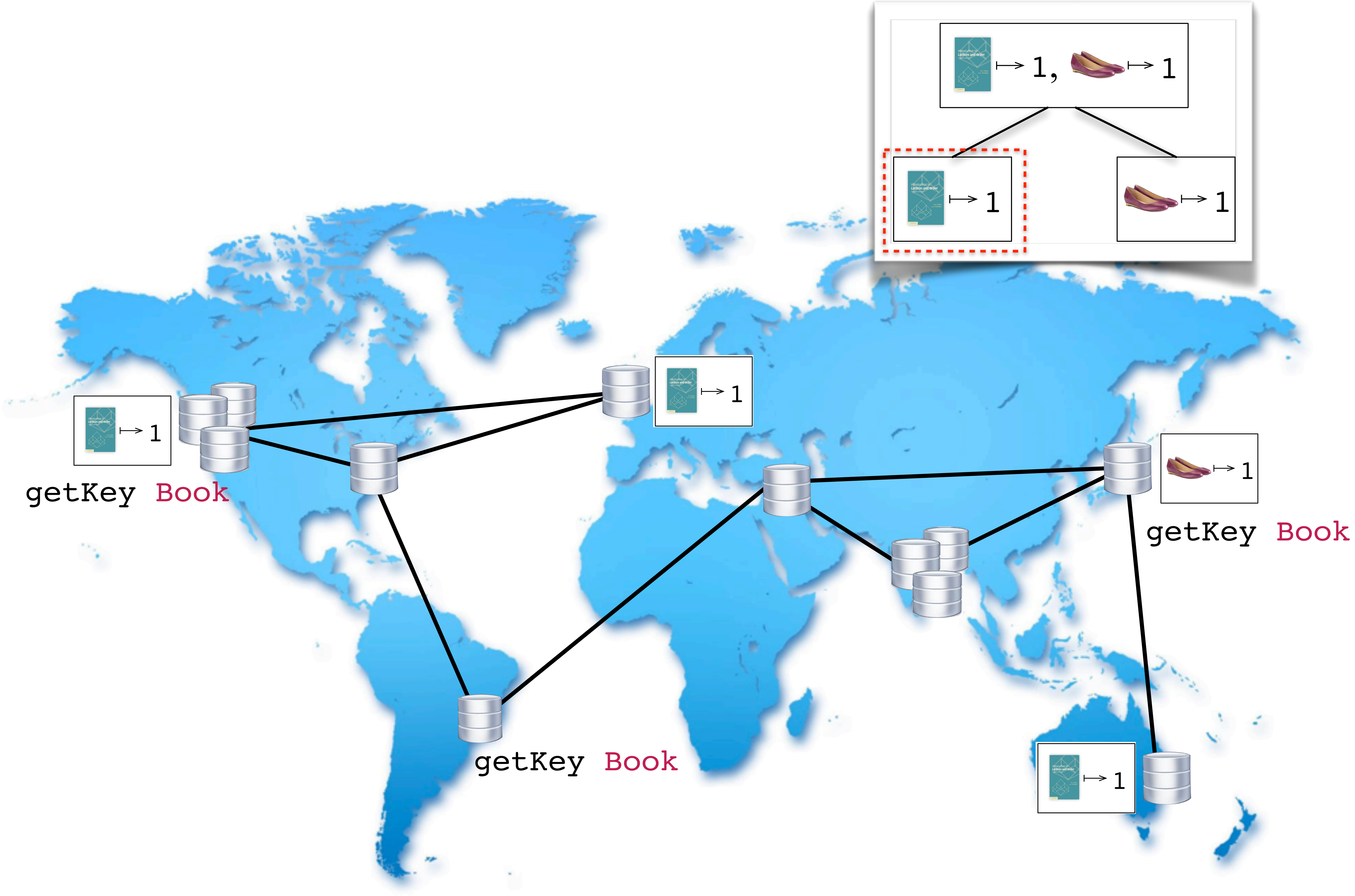
[DeCandia *et al.*, SOSP '07]



getKey Book

getKey Book

getKey Book



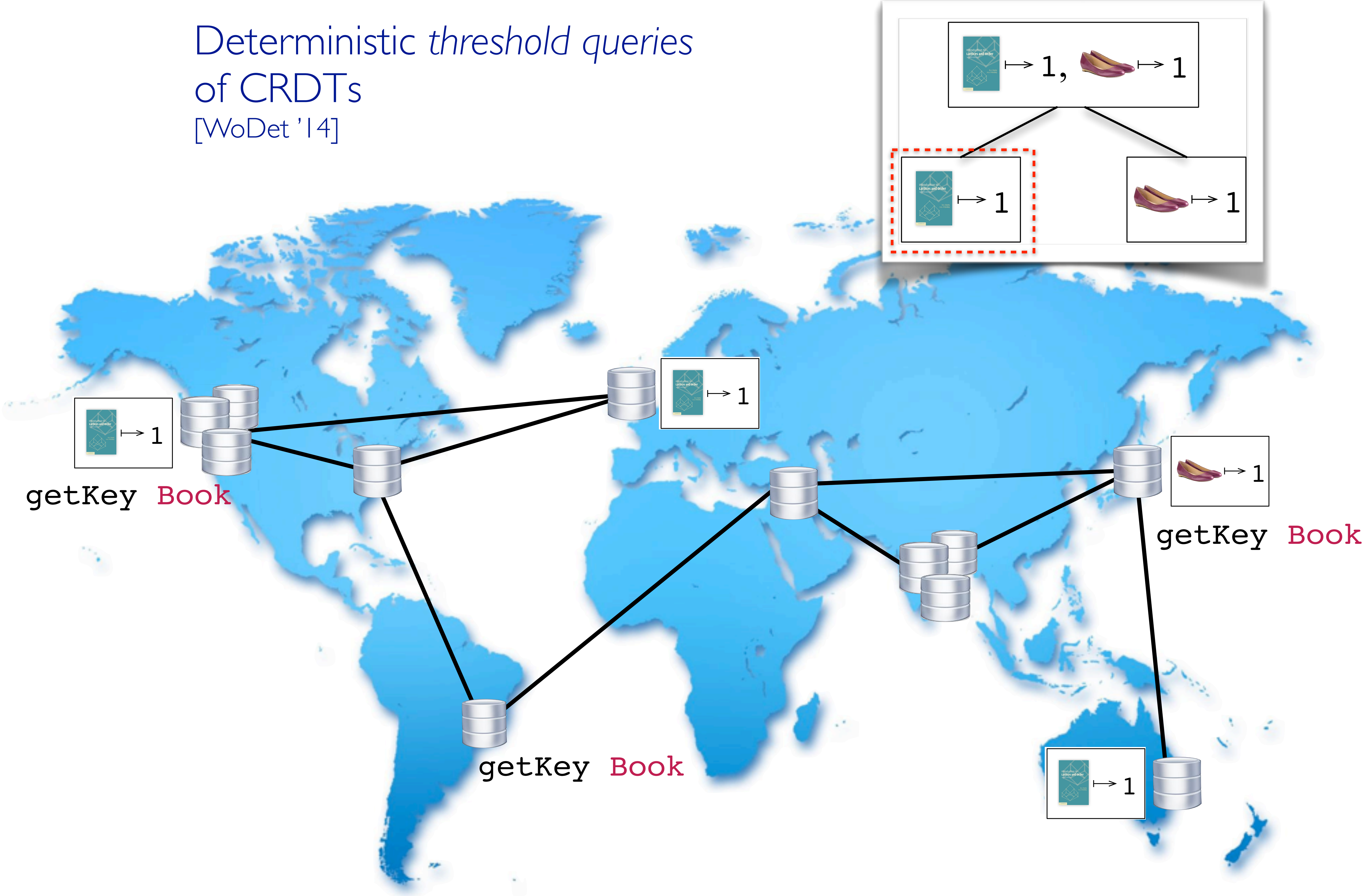
getKey Book

getKey Book

getKey Book

Deterministic *threshold queries* of CRDTs

[WoDet '14]



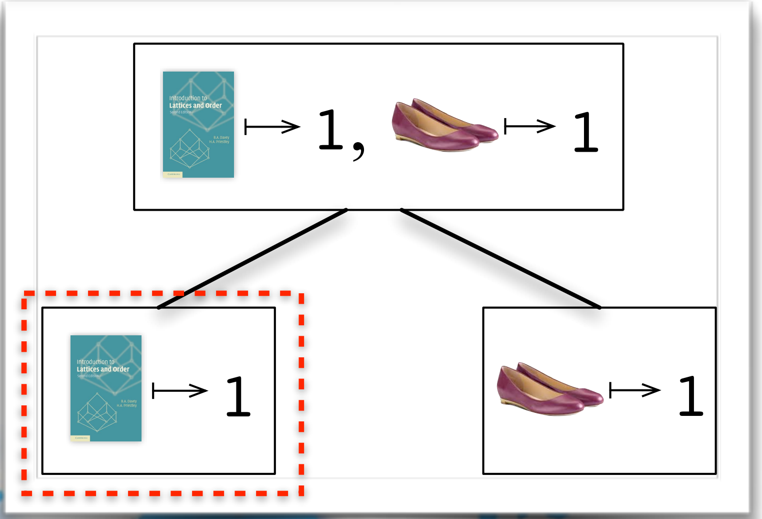
getKey Book

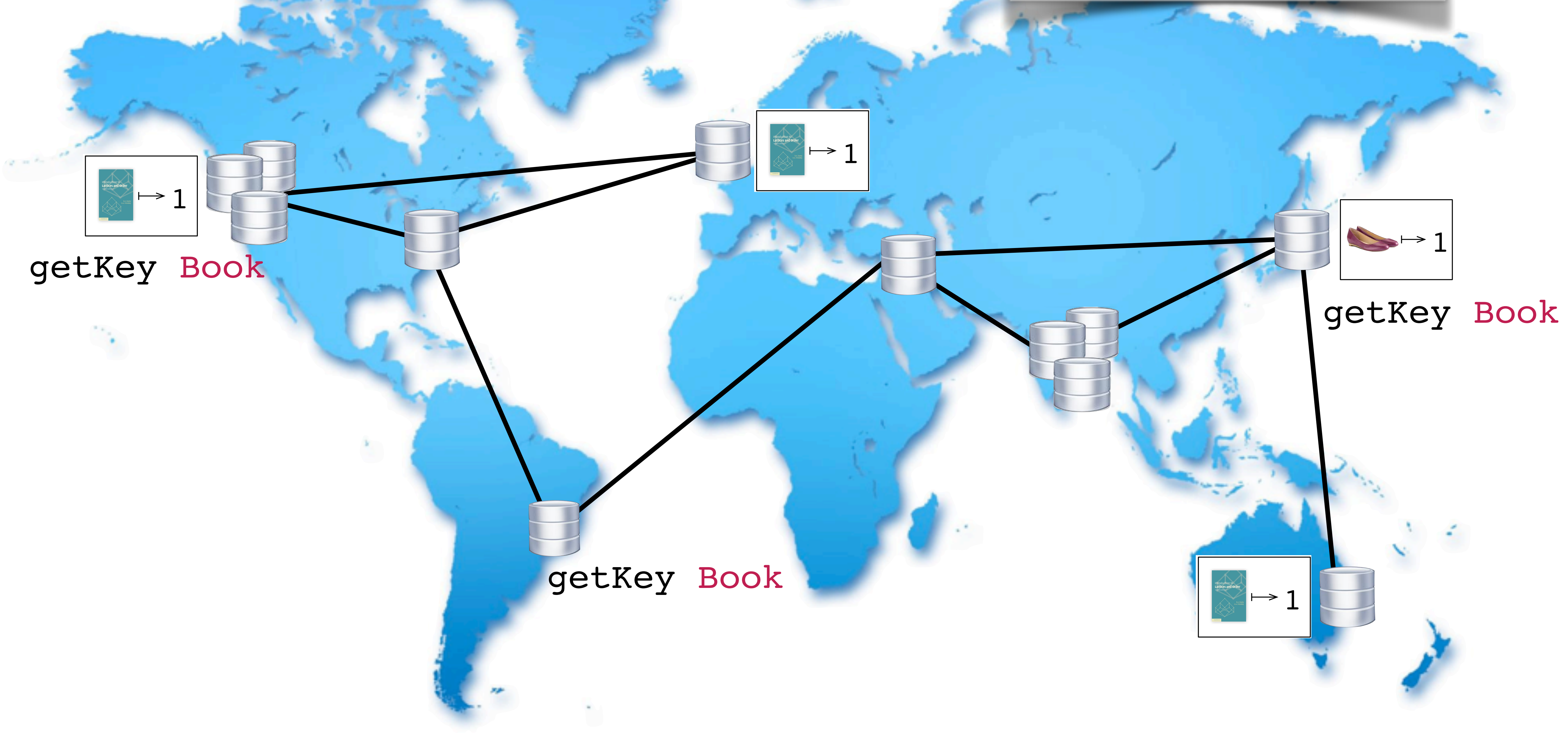
getKey Book

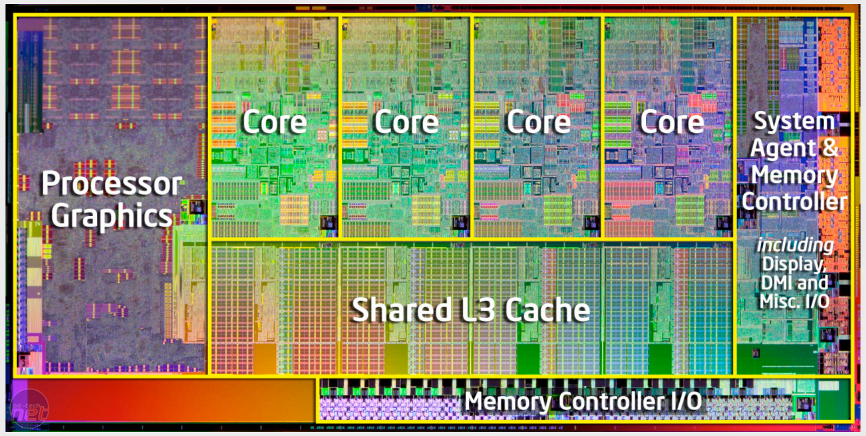
getKey Book

Deterministic *threshold queries* of CRDTs

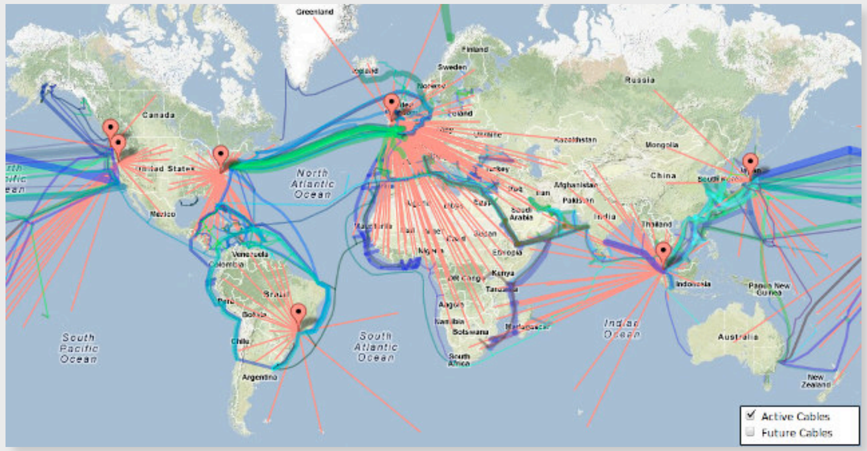
[WoDet '14]





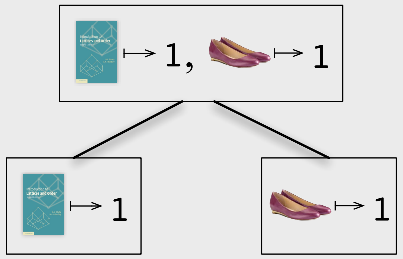


Parallel systems



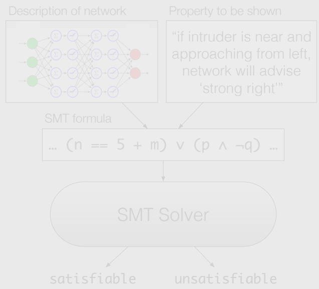
Distributed systems

language-based approaches

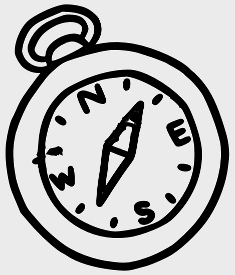


Lattice-based data structures (LVars) for deterministic programming
[POPL '14, PLDI '14, FHPC '13, WoDet '14]

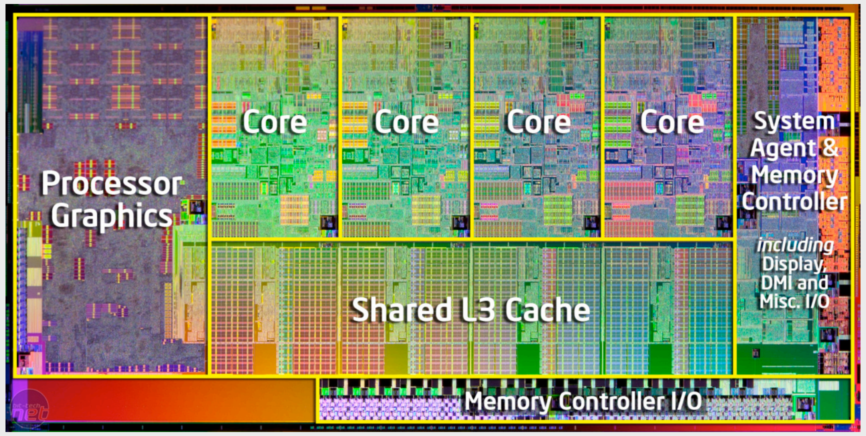
Non-invasive DSLs for productive parallelism
[ECOOP '17]

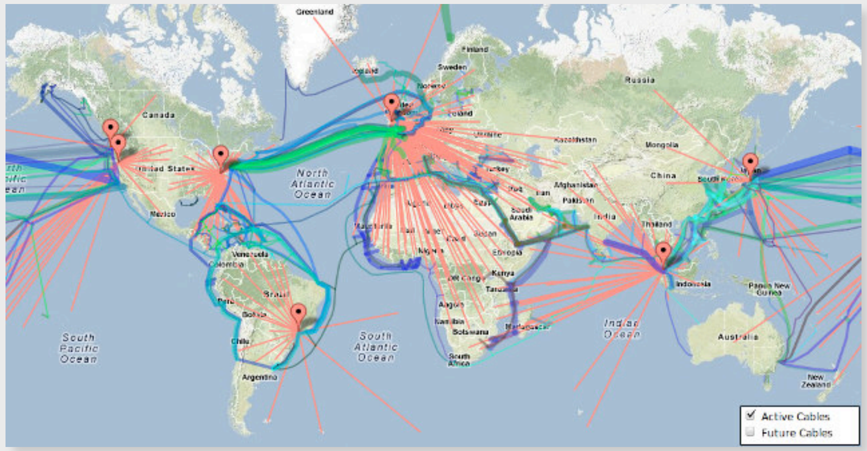

Description of network
Property to be shown
"if intruder is near and approaching from left, network will advise strong right"
SMT formula
... (n == 5 + m) ∨ (p ∧ ¬q) ...
SMT Solver
satisfiable
unsatisfiable

SMT-based verification of neural networks
[SysML '18]



Guiding principle:
Find the right **high-level abstractions**
to enable **efficient** computation

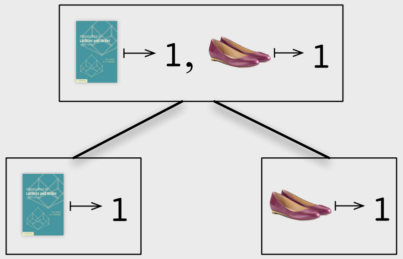




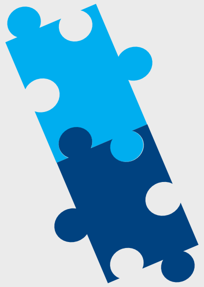
Parallel systems

Distributed systems

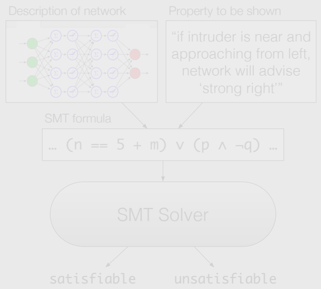
language-based approaches



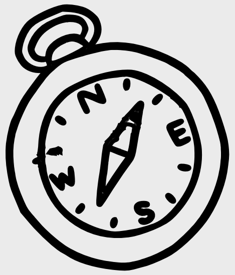
Lattice-based data structures (LVars) for deterministic programming
[POPL '14, PLDI '14, FHPC '13, WoDet '14]



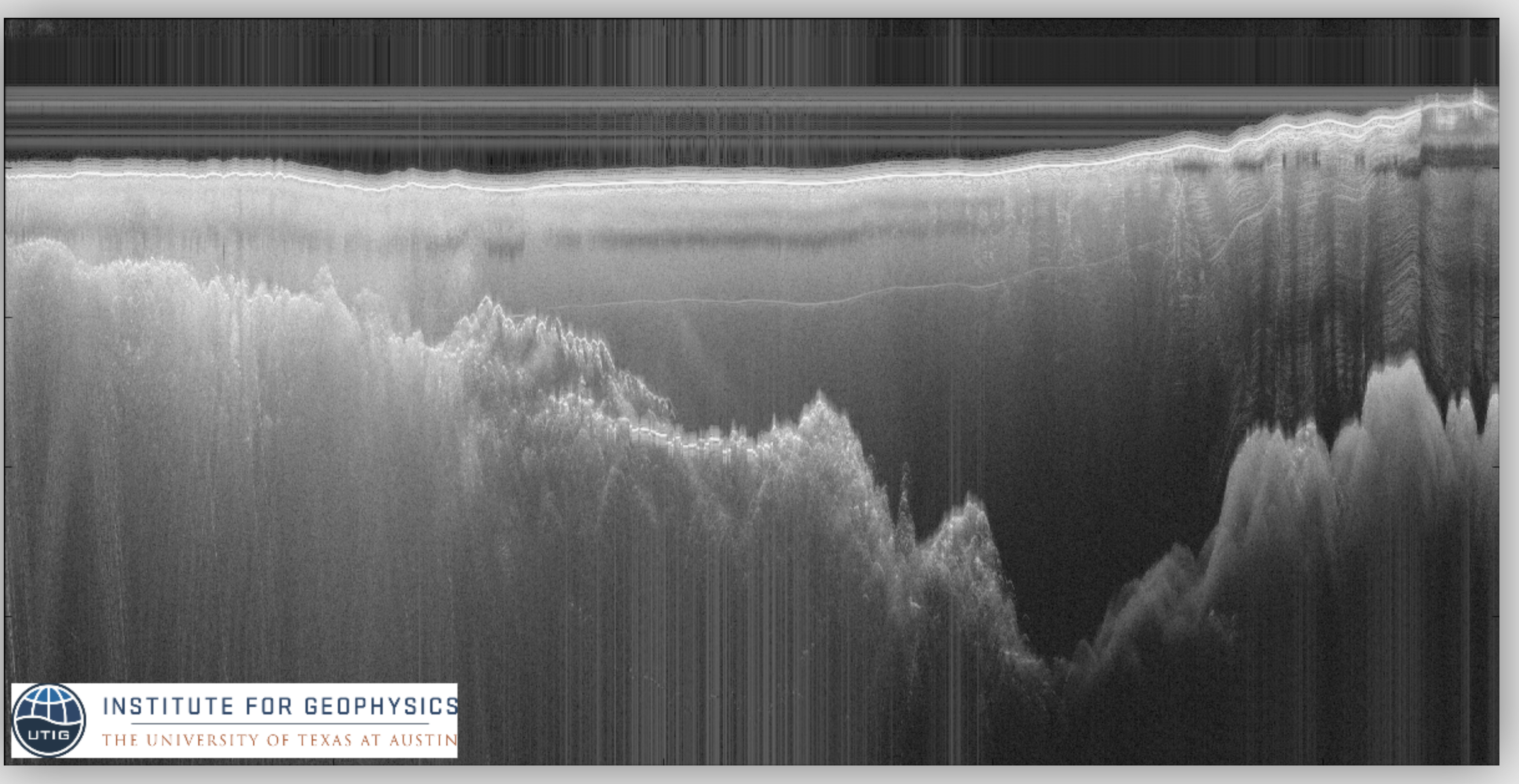
Non-invasive DSLs for productive parallelism
[ECOOP '17]


Description of network
Property to be shown
"If intruder is near and approaching from left, network will advise 'strong right'"
SMT formula
... (n == 5 + m) ∨ (p ∧ ¬q) ...
SMT Solver
satisfiable
unsatisfiable

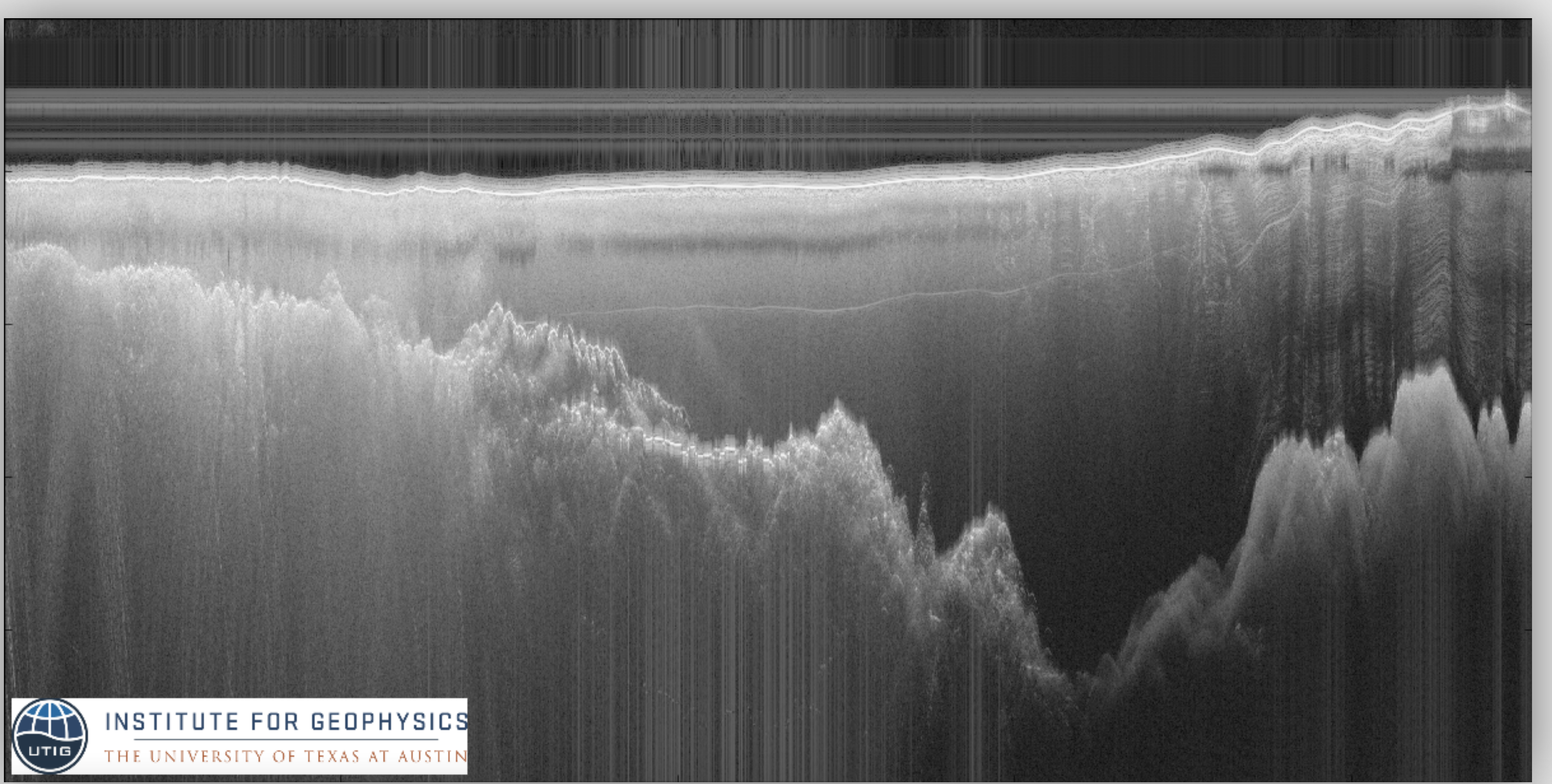
SMT-based verification of neural networks
[SysML '18]

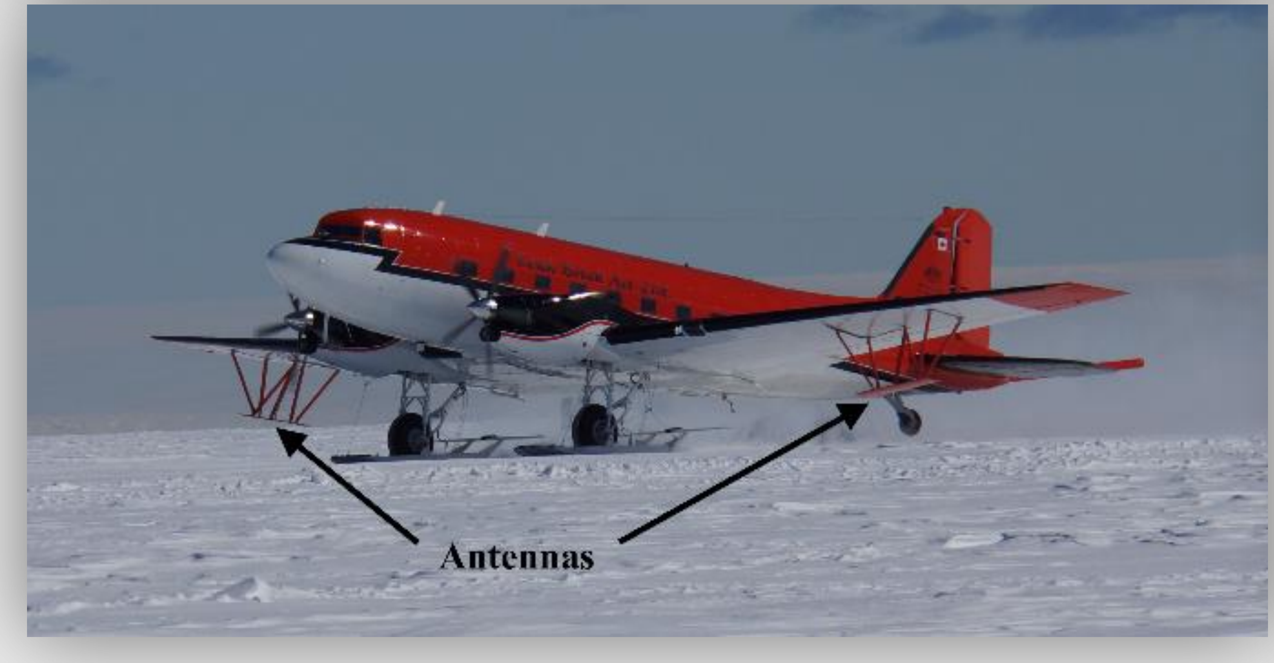


Guiding principle:
Find the right **high-level abstractions**
to enable **efficient** computation

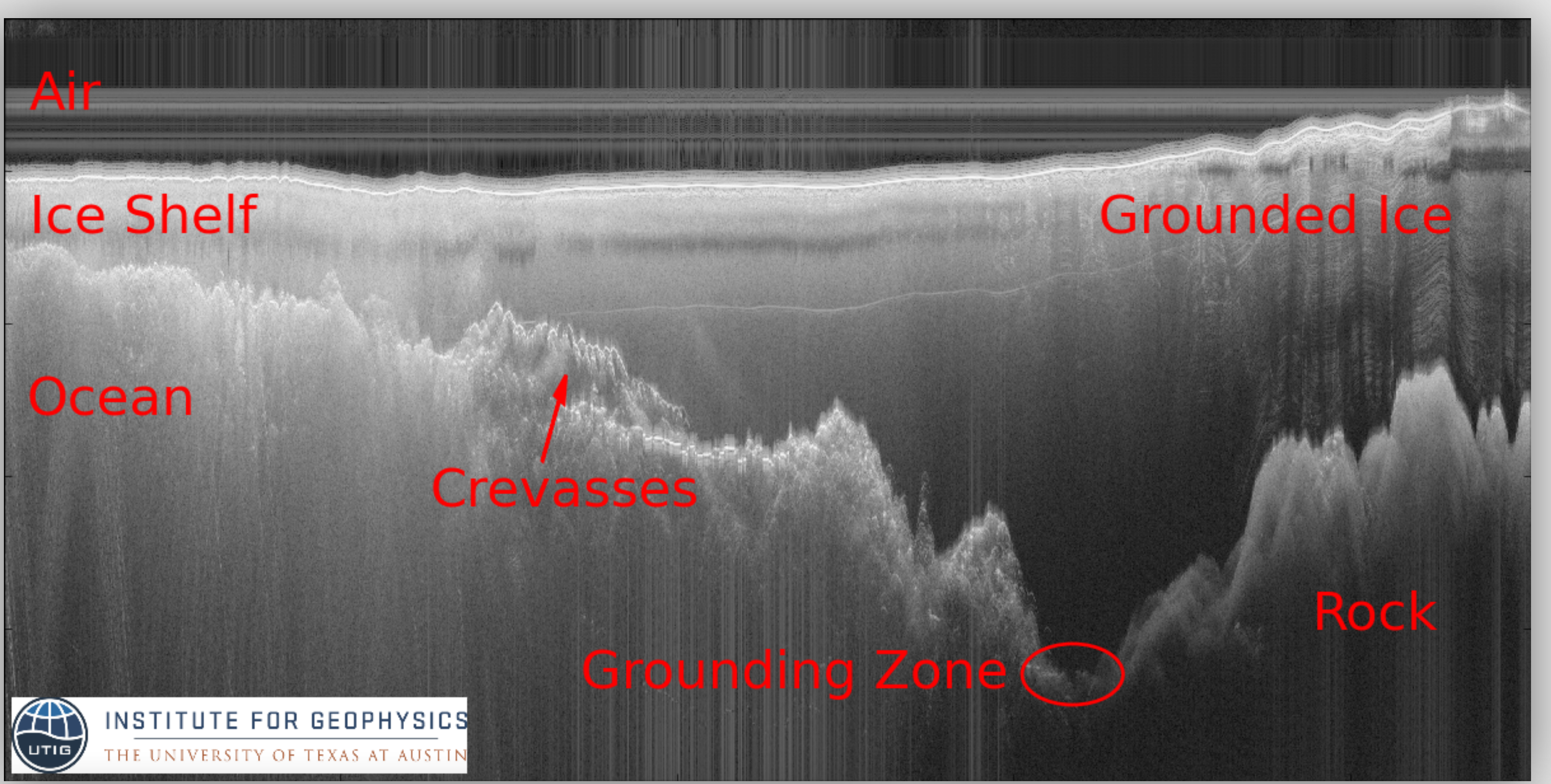


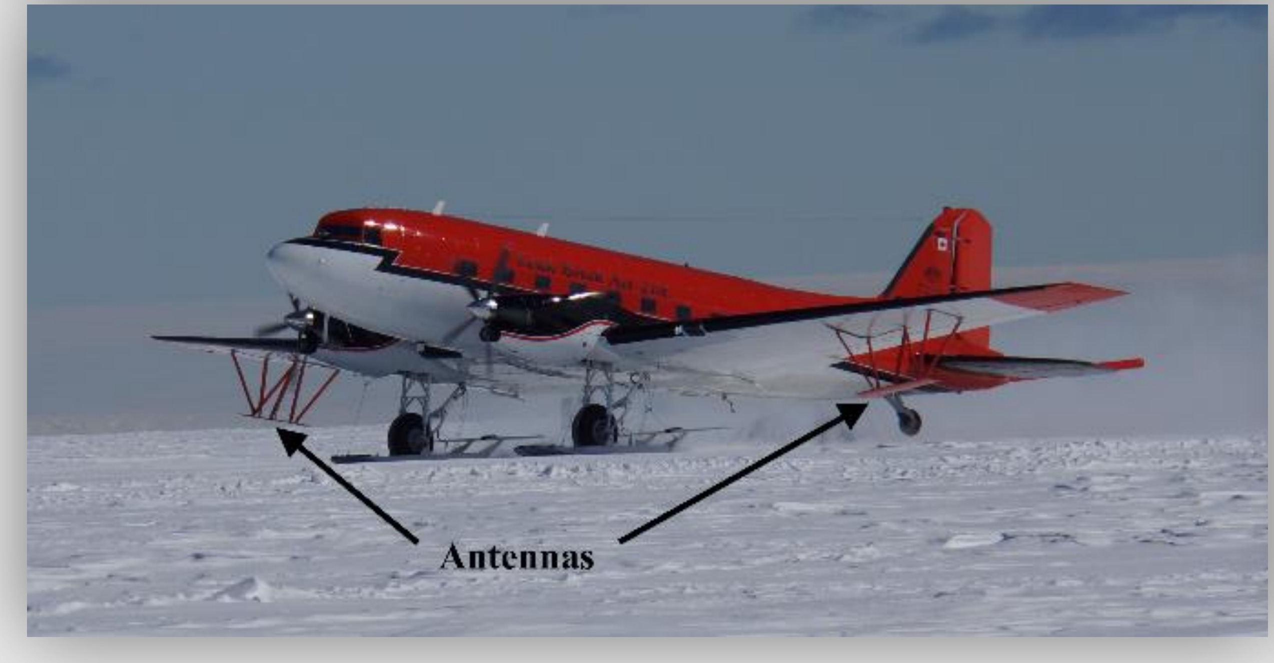
Source: http://lindzey.github.io/blog/2015/07/27/a-brief-introduction-to-ice-penetrating-radar/



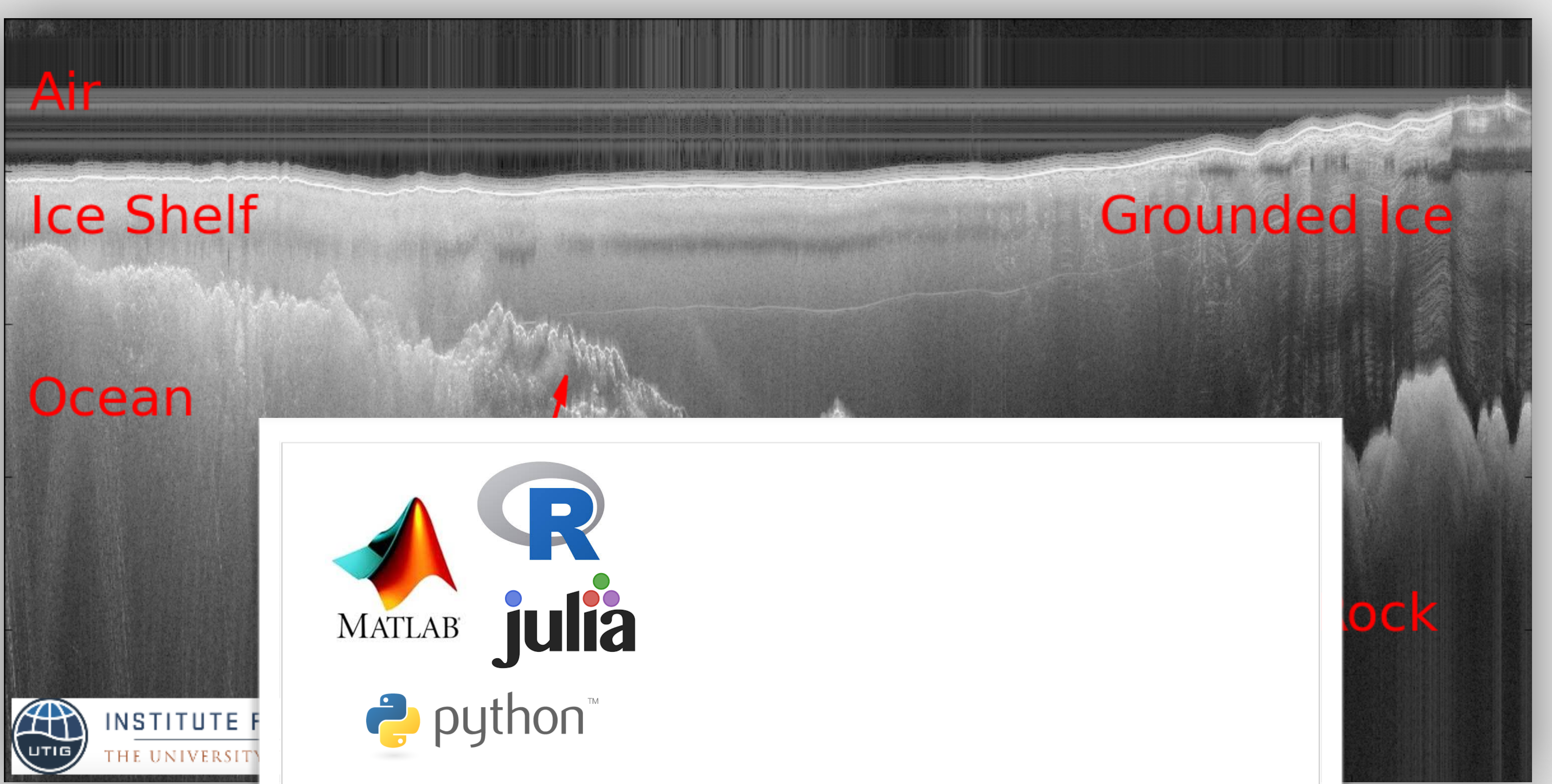


Source: http://lindzey.github.io/blog/2015/07/27/a-brief-introduction-to-ice-penetrating-radar/





Source: http://lindzey.github.io/blog/2015/07/27/a-brief-introduction-to-ice-penetrating-radar/

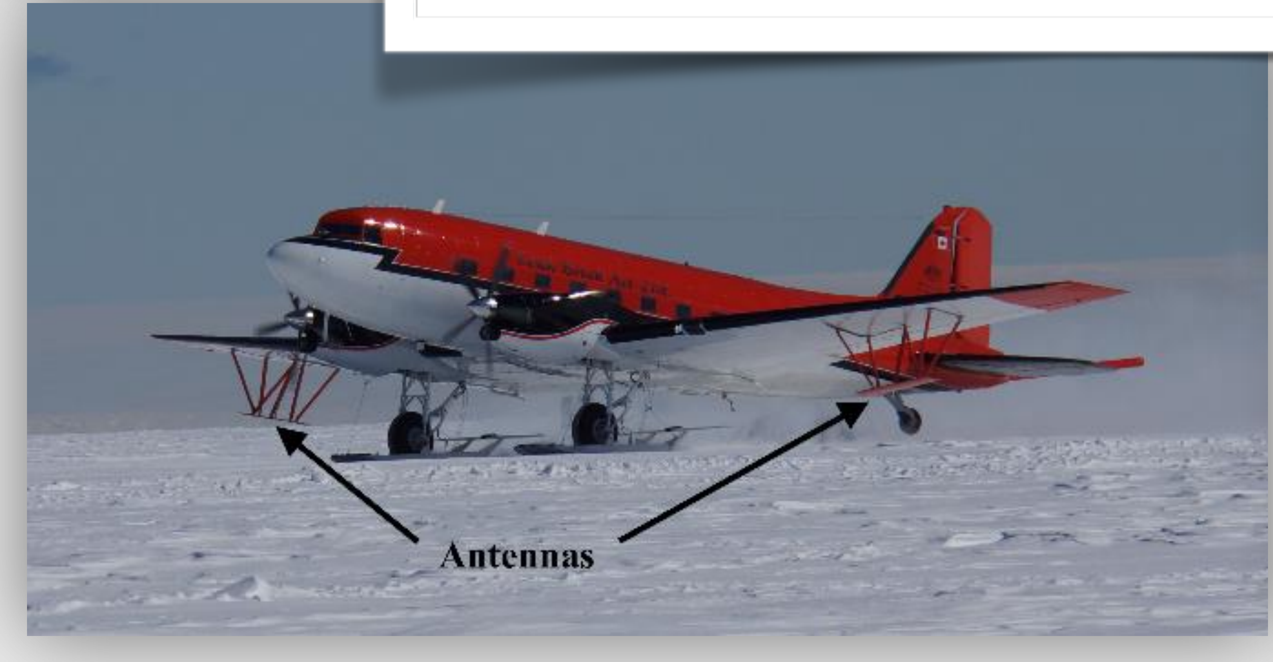


MATLAB

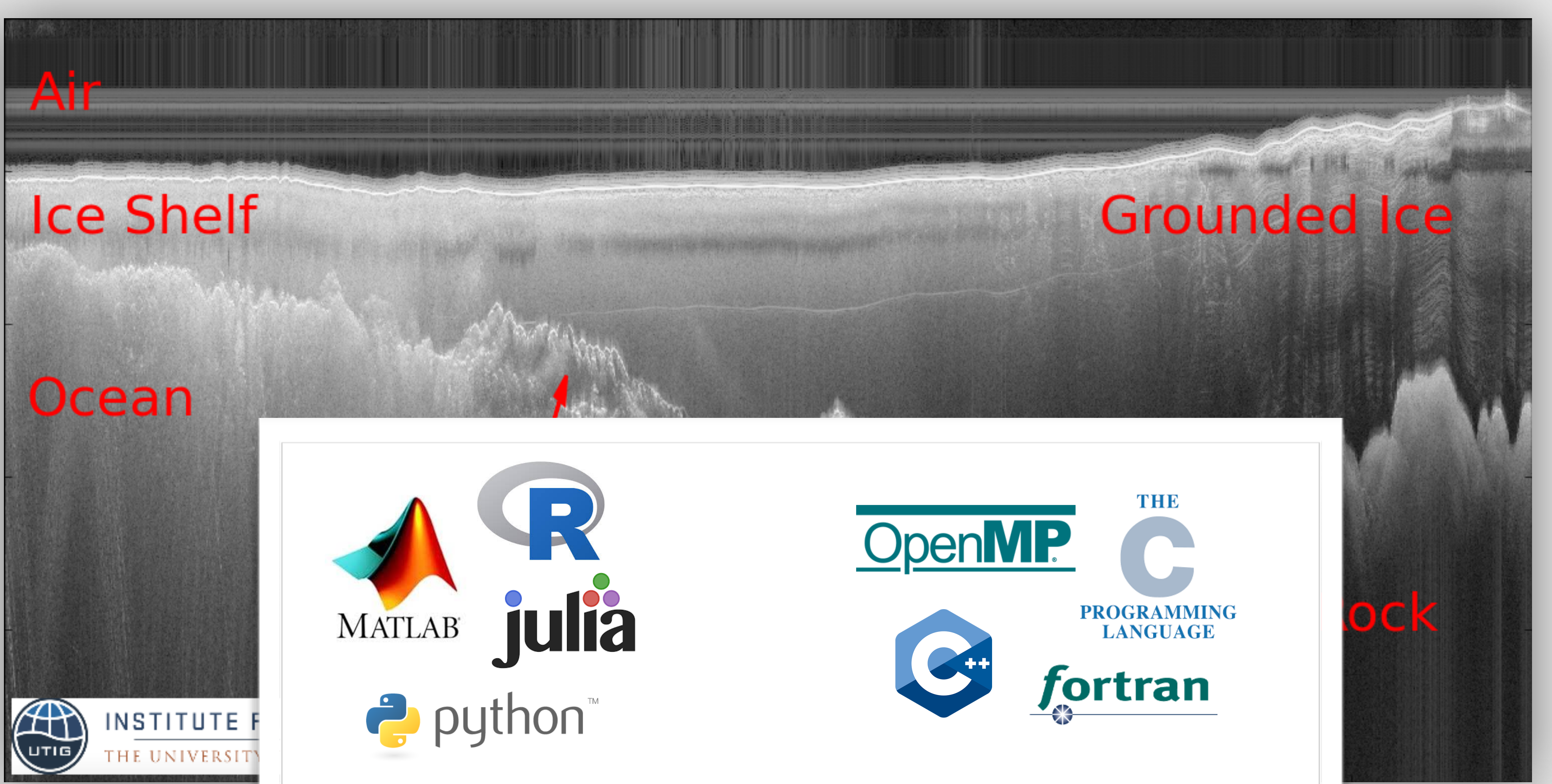
R

julia

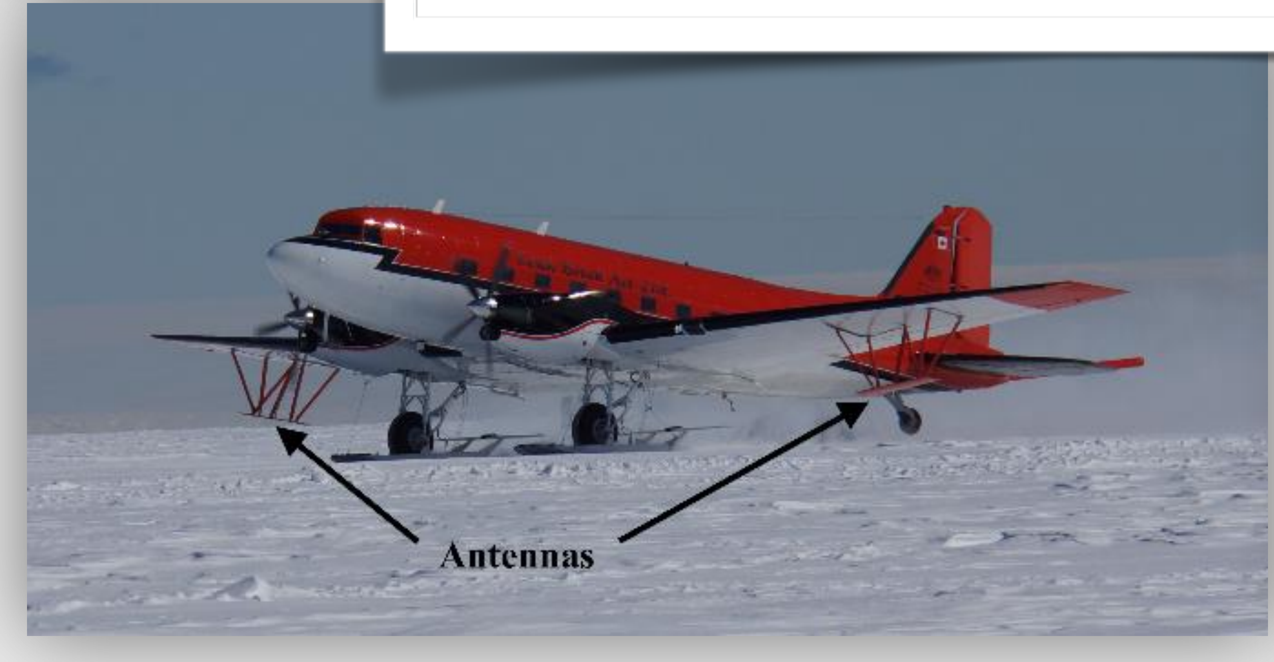
python™



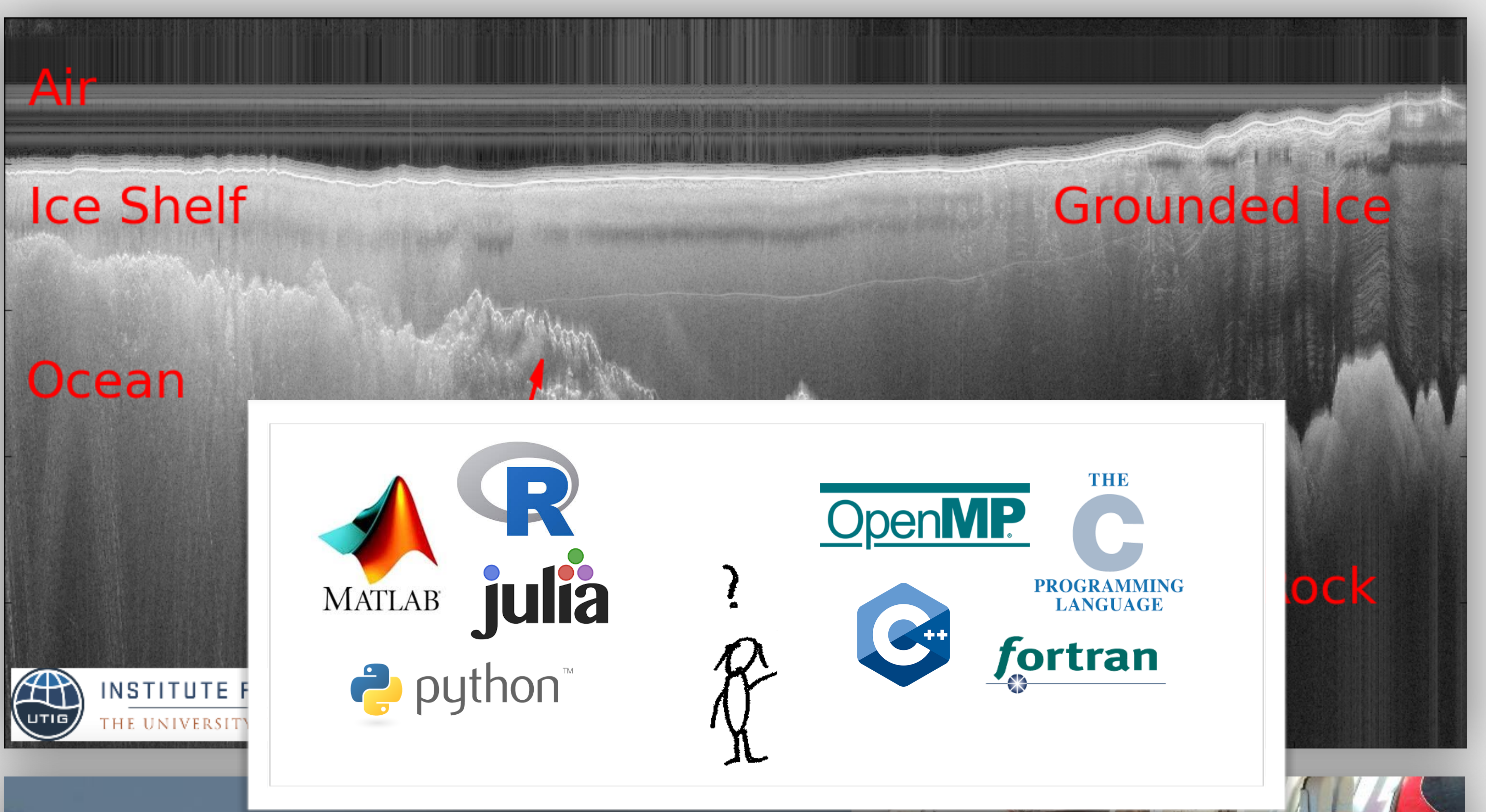
Source: http://lindzey.github.io/blog/2015/07/27/a-brief-introduction-to-ice-penetrating-radar/

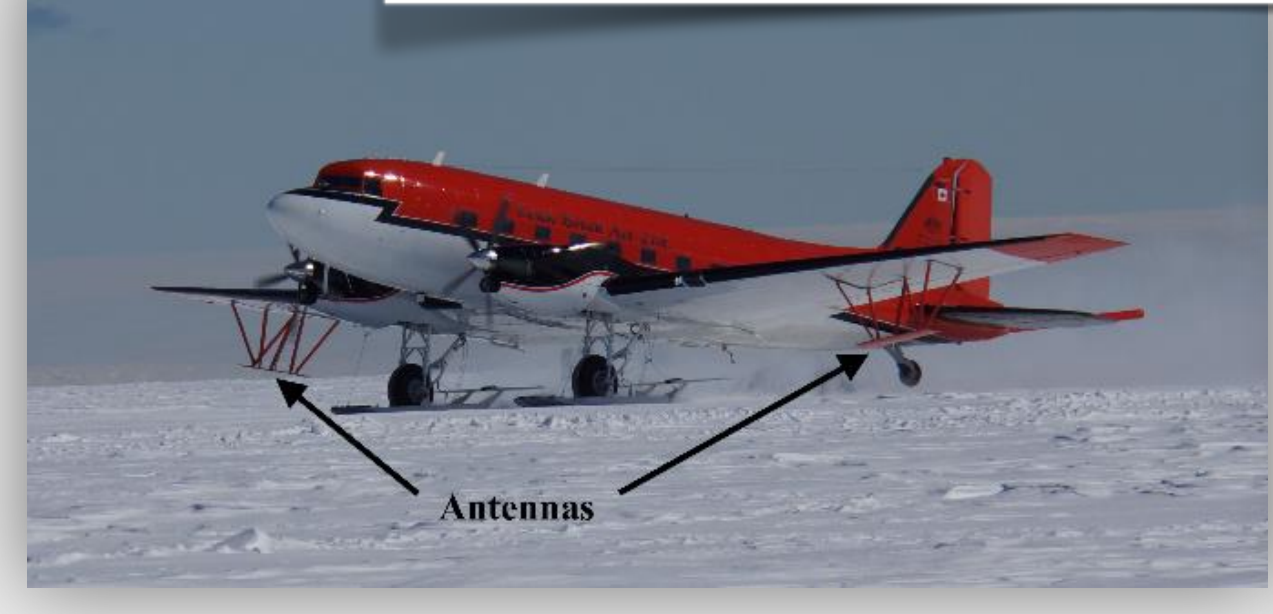


MATLAB
R
julia
python
OpenMP
THE C PROGRAMMING LANGUAGE
C++
fortran



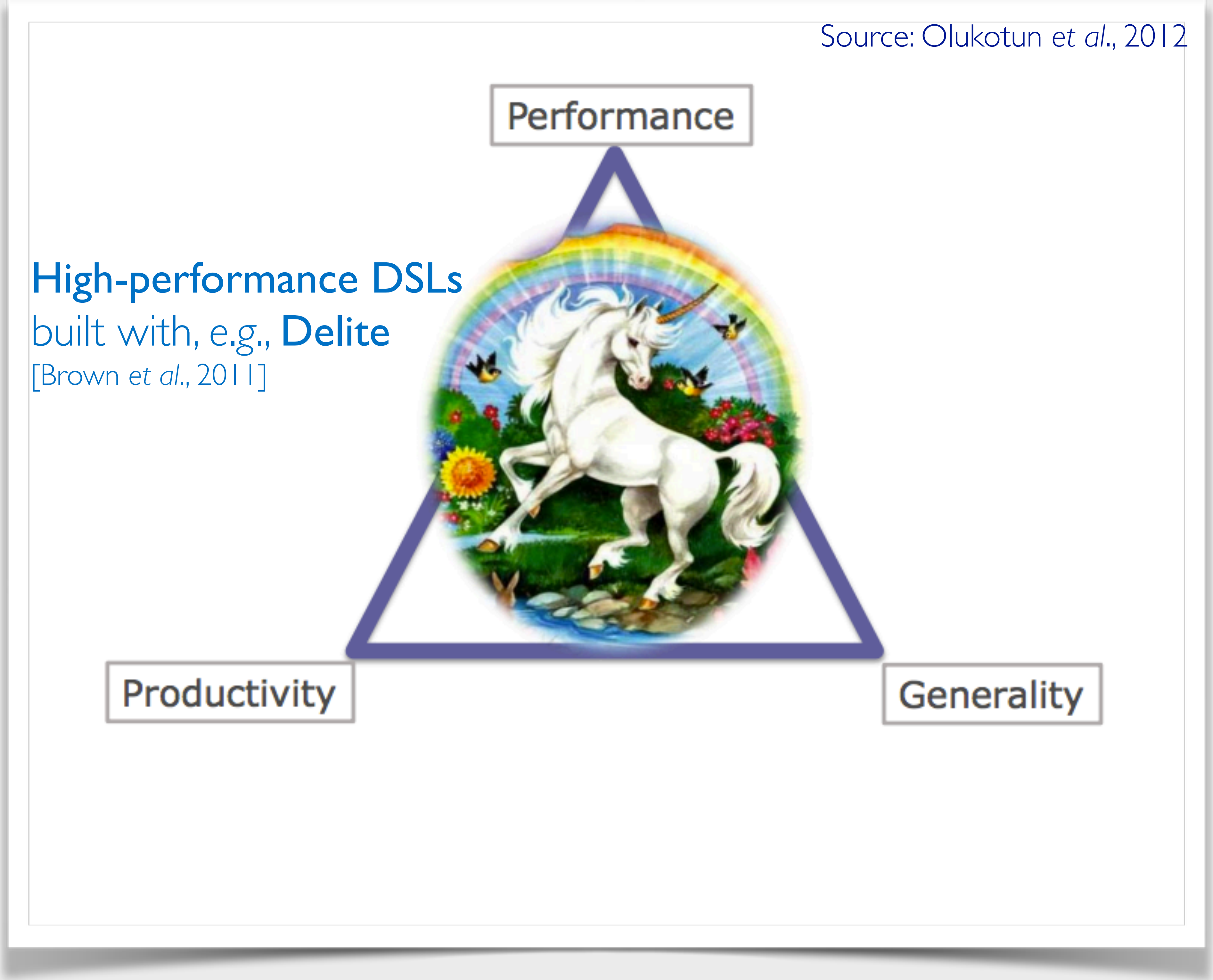
Source: http://lindzey.github.io/blog/2015/07/27/a-brief-introduction-to-ice-penetrating-radar/





Source: http://lindzey.github.io/blog/2015/07/27/a-brief-introduction-to-ice-penetrating-radar/

Source: Olukotun *et al.*, 2012

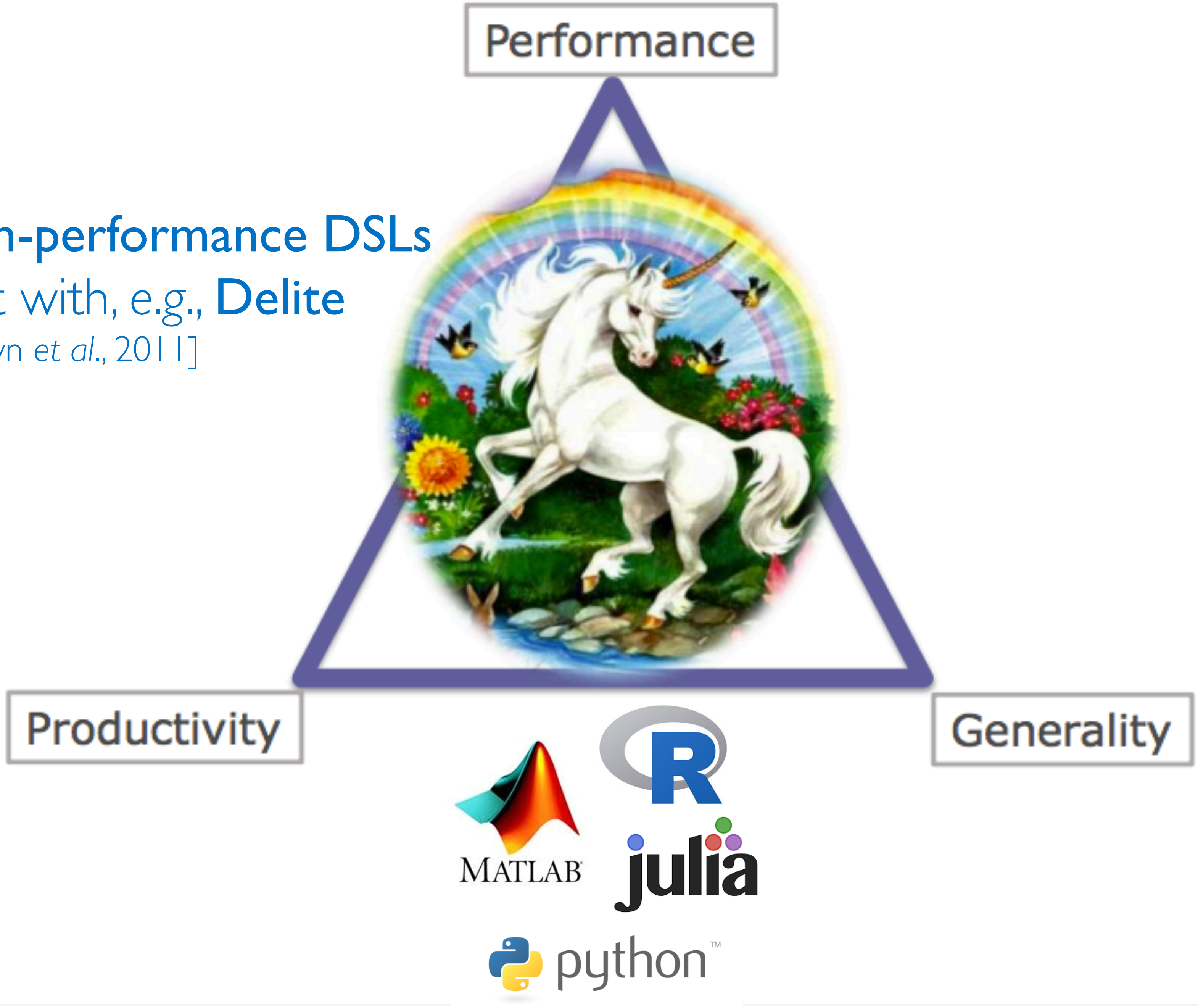


High-performance DSLs
built with, e.g., **Delite**
[Brown *et al.*, 2011]

Sacrifice **generality** for **productivity** and **performance**?

Source: Olukotun *et al.*, 2012

High-performance DSLs
built with, e.g., **Delite**
[Brown *et al.*, 2011]

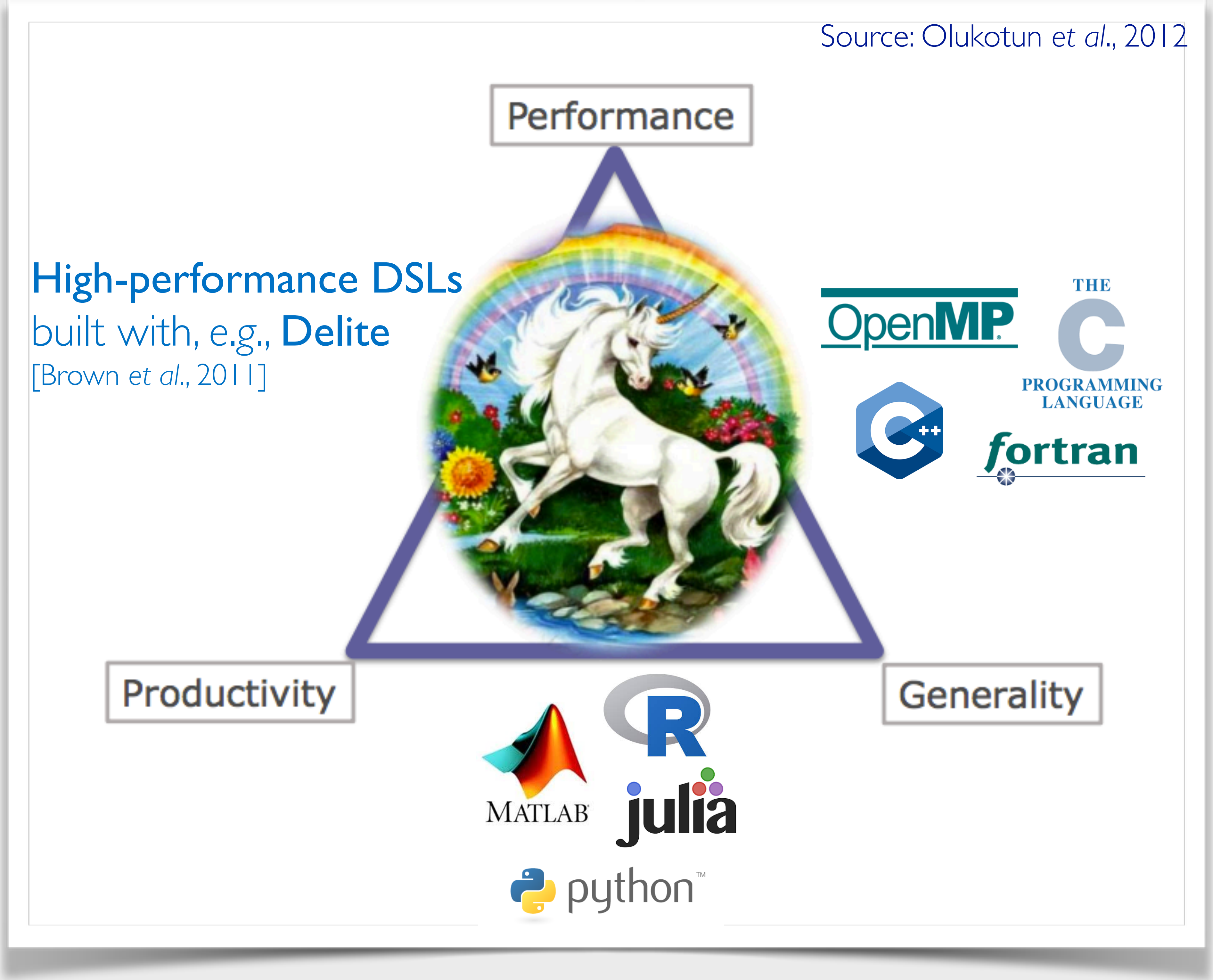


Performance

Productivity

Generality

Sacrifice **generality** for **productivity** and **performance**?



Source: Olukotun *et al.*, 2012

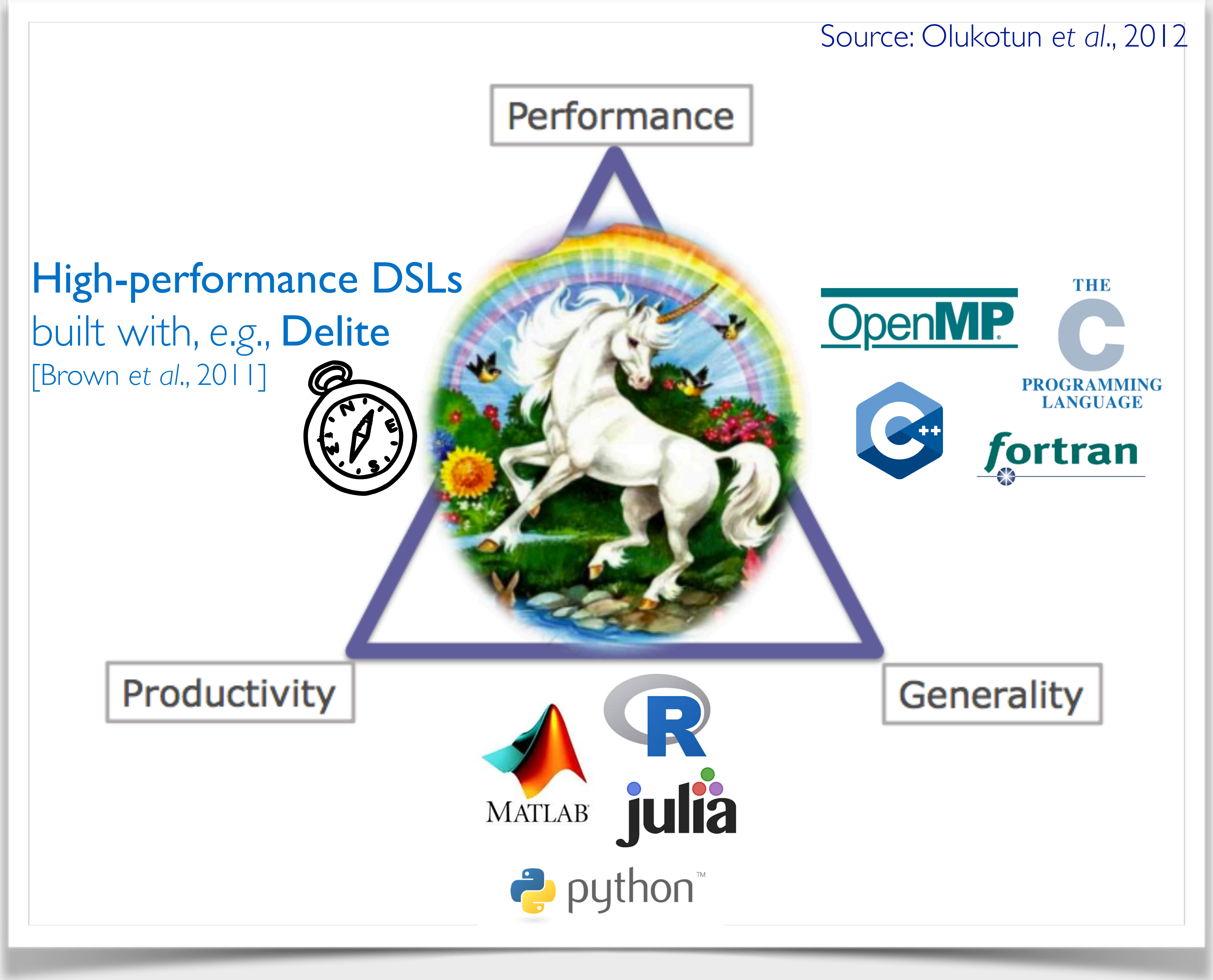
Performance

High-performance DSLs
built with, e.g., **Delite**
[Brown *et al.*, 2011]

Productivity

Generality

Sacrifice **generality** for **productivity** and **performance**?



Source: Olukotun *et al.*, 2012

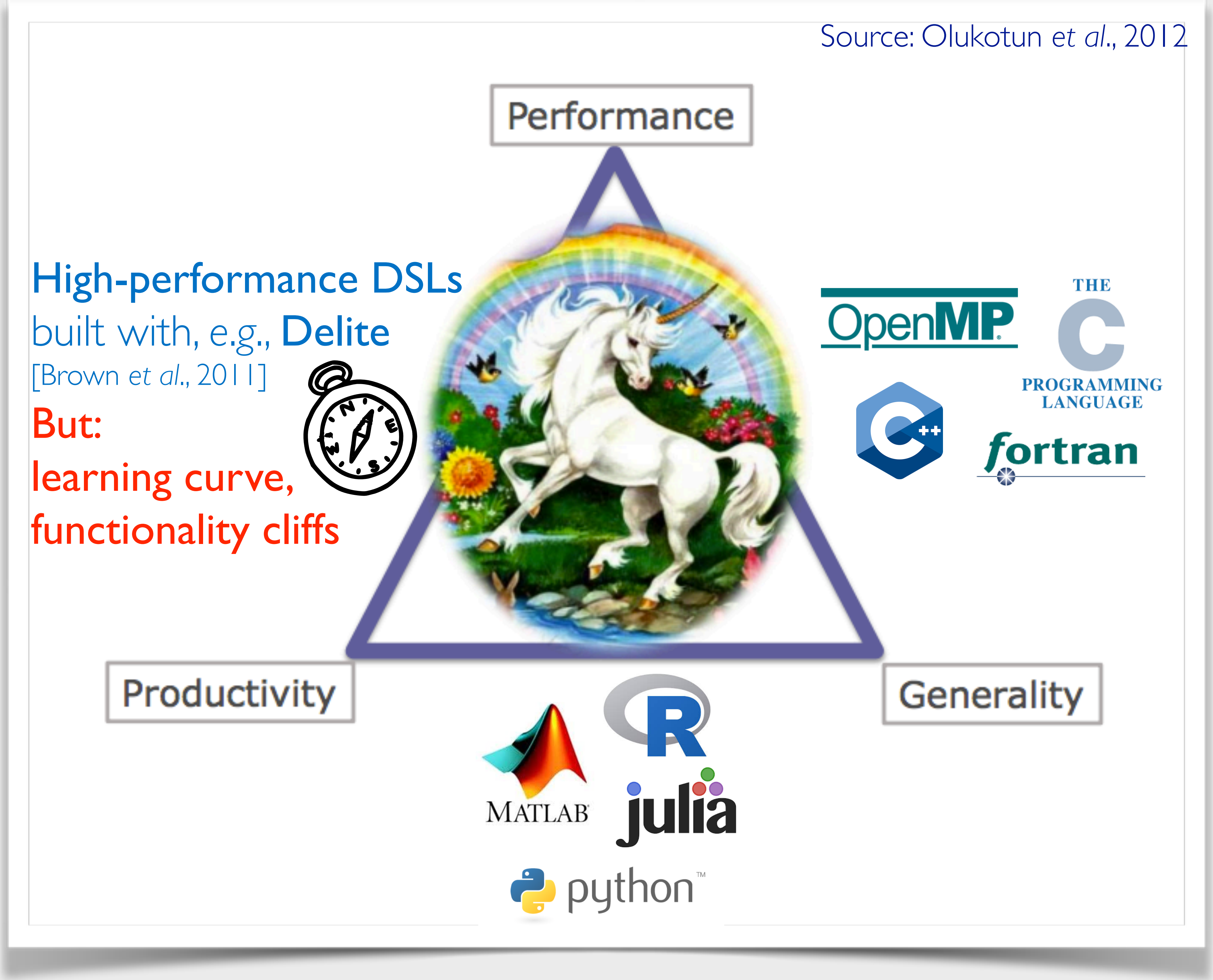
Performance

Productivity

Generality

High-performance DSLs
built with, e.g., **Delite**
[Brown *et al.*, 2011]

Sacrifice **generality** for **productivity** and **performance**?



Source: Olukotun *et al.*, 2012

Performance

High-performance DSLs
built with, e.g., **Delite**
[Brown *et al.*, 2011]

But:
learning curve,
functionality cliffs

Productivity

Generality

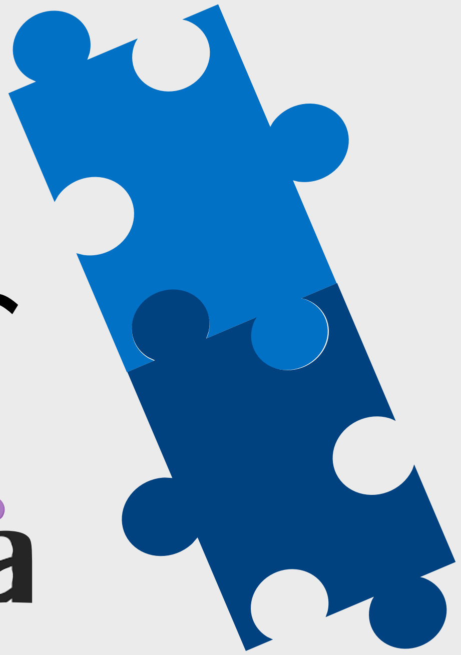
Sacrifice **generality** for **productivity** and **performance**?

ParallelAccelerator

[ECOOP '17]

ParallelAccelerator

[ECOOP '17]



A **non-invasive DSL** embedded in julia

Accelerates existing language constructs with `@acc`

Supports additional `runStencil` construct

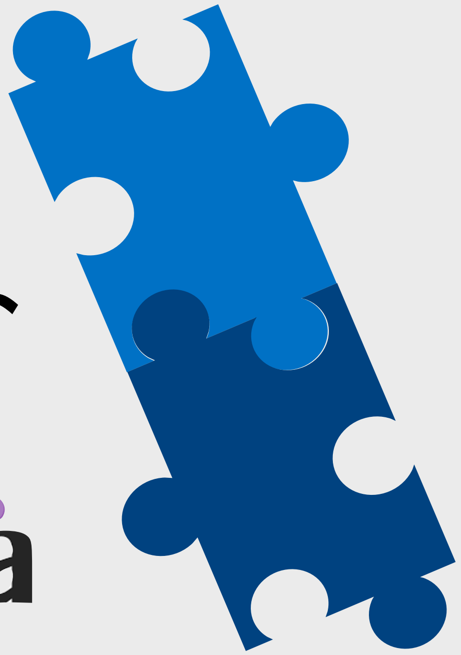
A **combination compiler-library** solution

Library mode for development and debugging

Native mode for high performance at deployment

ParallelAccelerator

[ECOOP '17]



A **non-invasive DSL** embedded in julia

Accelerates existing language constructs with `@acc`

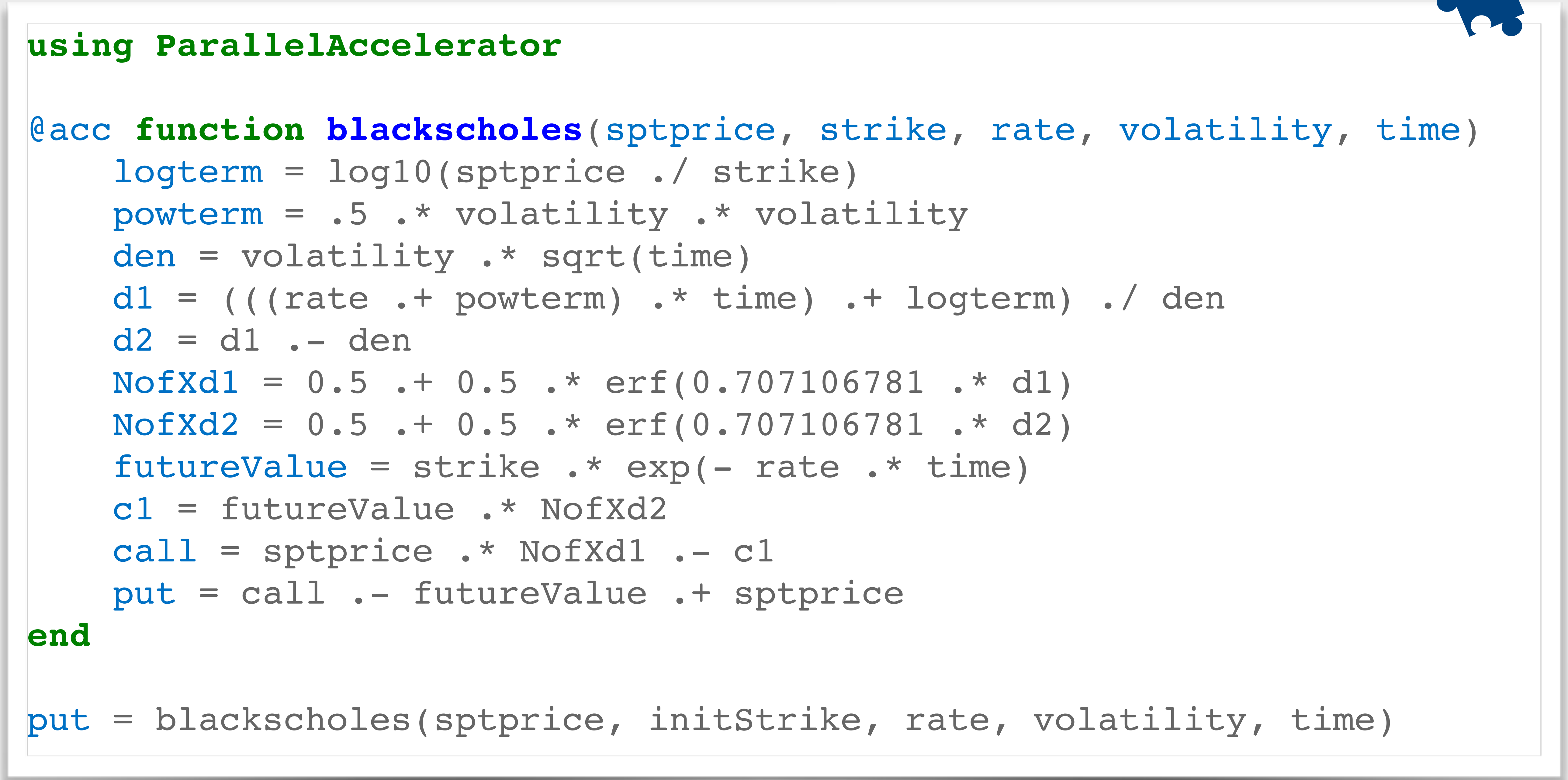
Supports additional `runStencil` construct

A **combination compiler-library** solution

- Library mode for development and debugging
- Native mode for high performance at deployment

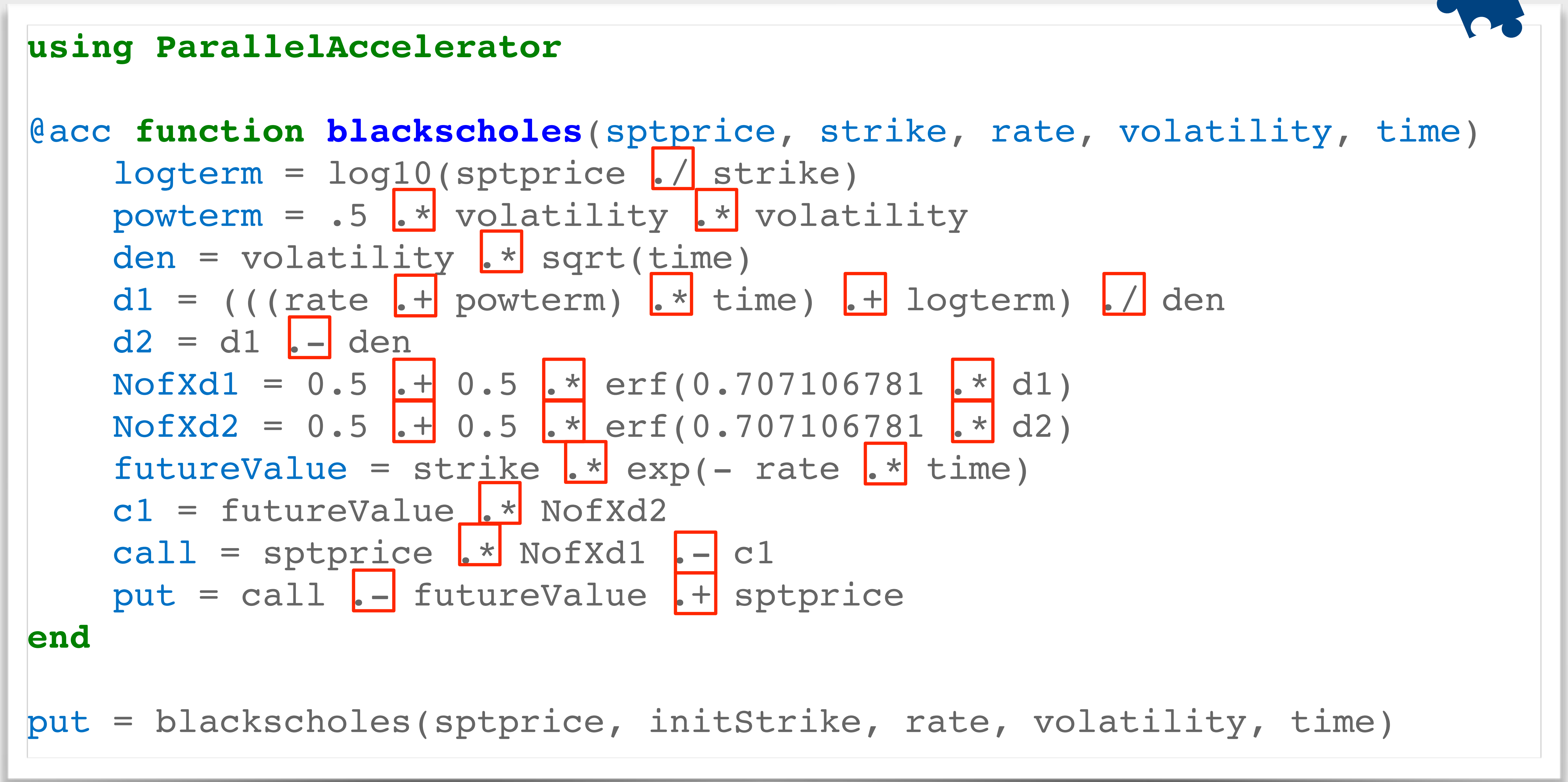
`github.com/IntelLabs/ParallelAccelerator.jl`

@acc example: Black-Scholes option pricing



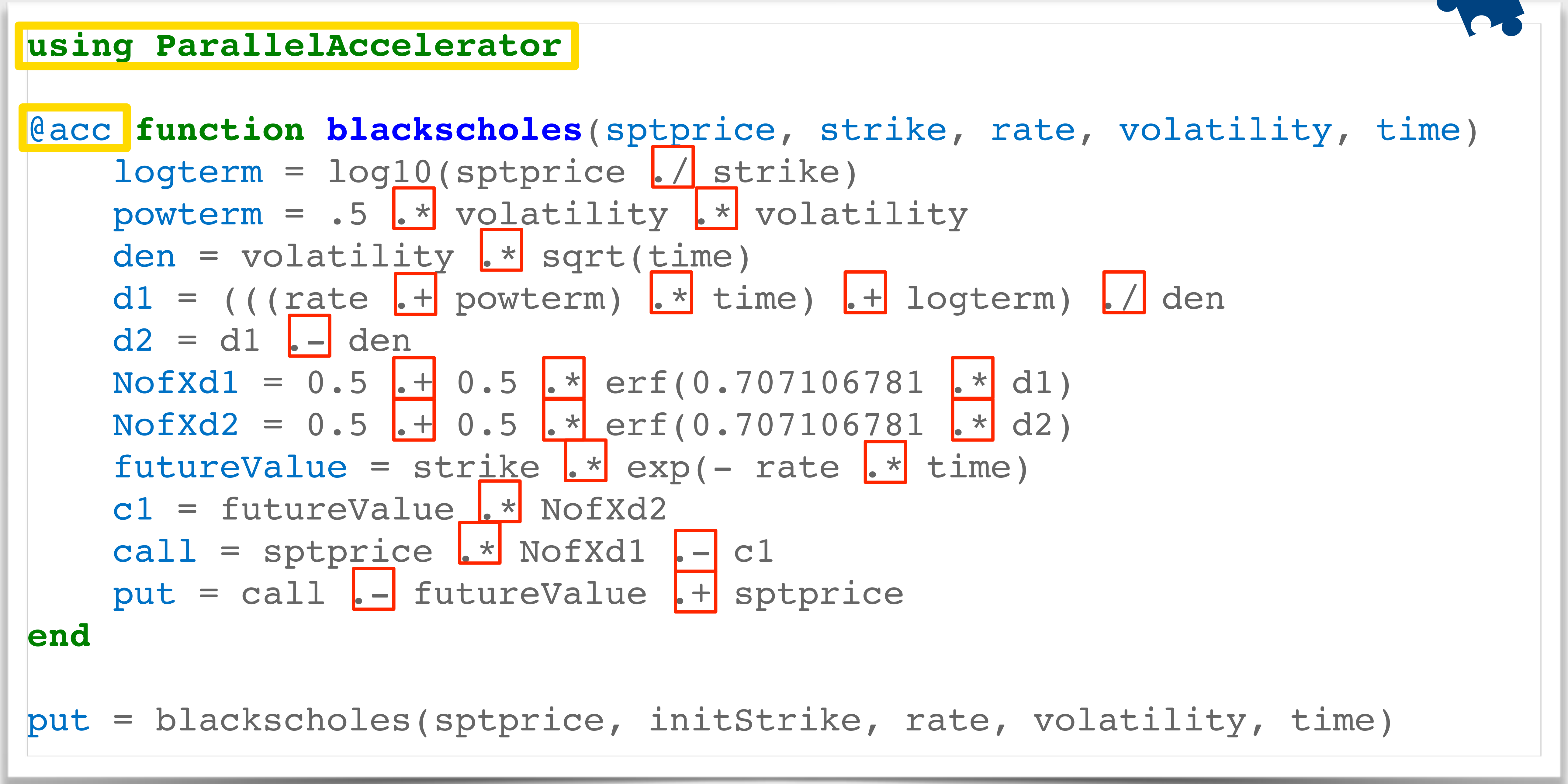
```
using ParallelAccelerator

@acc function blackscholes(sptprice, strike, rate, volatility, time)
    logterm = log10(sptprice ./ strike)
    powterm = .5 .* volatility .* volatility
    den = volatility .* sqrt(time)
    d1 = (((rate .+ powterm) .* time) .+ logterm) ./ den
    d2 = d1 .- den
    NofXd1 = 0.5 .+ 0.5 .* erf(0.707106781 .* d1)
    NofXd2 = 0.5 .+ 0.5 .* erf(0.707106781 .* d2)
    futureValue = strike .* exp(- rate .* time)
    c1 = futureValue .* NofXd2
    call = sptprice .* NofXd1 .- c1
    put = call .- futureValue .+ sptprice
end

put = blackscholes(sptprice, initStrike, rate, volatility, time)
```

`@acc` example: Black-Scholes option pricing



```
using ParallelAccelerator

@acc function blackscholes(sptprice, strike, rate, volatility, time)
    logterm = log10(sptprice ./ strike)
    powterm = .5 .* volatility .* volatility
    den = volatility .* sqrt(time)
    d1 = (((rate .+ powterm) .* time) .+ logterm) ./ den
    d2 = d1 .- den
    NofXd1 = 0.5 .+ 0.5 .* erf(0.707106781 .* d1)
    NofXd2 = 0.5 .+ 0.5 .* erf(0.707106781 .* d2)
    futureValue = strike .* exp(- rate .* time)
    c1 = futureValue .* NofXd2
    call = sptprice .* NofXd1 .- c1
    put = call .- futureValue .+ sptprice
end

put = blackscholes(sptprice, initStrike, rate, volatility, time)
```

@acc example: Black-Scholes option pricing



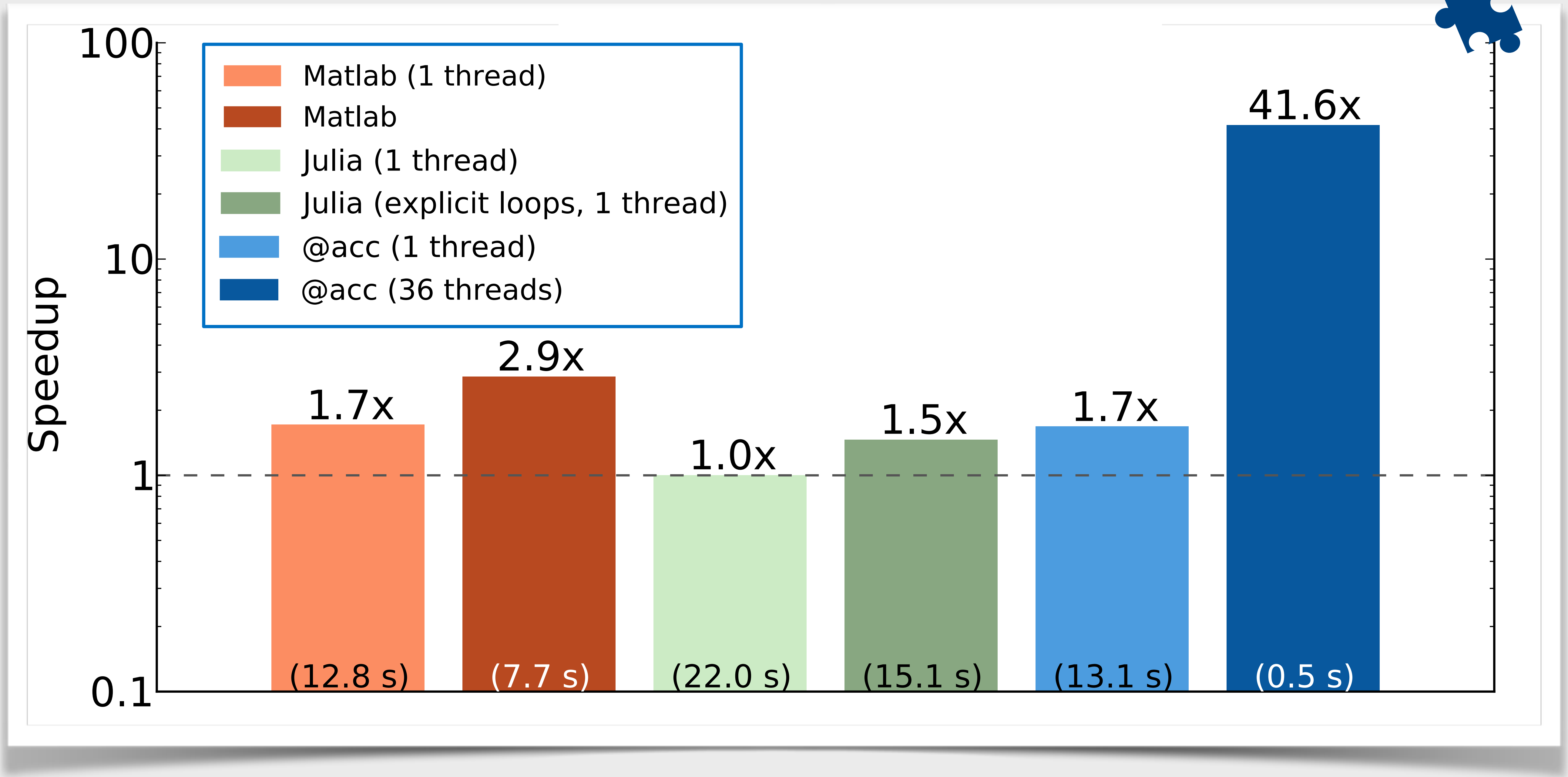
```
using ParallelAccelerator

@acc function blackscholes(sptprice, strike, rate, volatility, time)
    logterm = log10(sptprice ./ strike)
    powterm = .5 .* volatility .* volatility
    den = volatility .* sqrt(time)
    d1 = (((rate .+ powterm) .* time) .+ logterm) ./ den
    d2 = d1 .- den
    NofXd1 = 0.5 .+ 0.5 .* erf(0.707106781 .* d1)
    NofXd2 = 0.5 .+ 0.5 .* erf(0.707106781 .* d2)
    futureValue = strike .* exp(- rate .* time)
    c1 = futureValue .* NofXd2
    call = sptprice .* NofXd1 .- c1
    put = call .- futureValue .+ sptprice
end

put = blackscholes(sptprice, initStrike, rate, volatility, time)
```

@acc example: Black-Scholes option pricing

```
using ParallelAccelerator

@acc function blackscholes(sptprice, strike, rate, volatility, time)
    logterm = log10(sptprice ./ strike)
    powterm = .5 .* volatility .* volatility
    den = volatility .* sqrt(time)
    d1 = (((rate .+ powterm) .* time) .+ logterm) ./ den
    d2 = d1 .- den
    NofXd1 = 0.5 .+ 0.5 .* erf(0.707106781 .* d1)
    NofXd2 = 0.5 .+ 0.5 .* erf(0.707106781 .* d2)
    futureValue = strike .* exp(- rate .* time)
    c1 = futureValue .* NofXd2
    call = sptprice .* NofXd1 .- c1
    put = call .- futureValue .+ sptpri
end

put = blackscholes(sptprice, initStrike
```

Translates pointwise array operations like `.+`, `.-`, `.*`, and `./` to **data-parallel map** operations

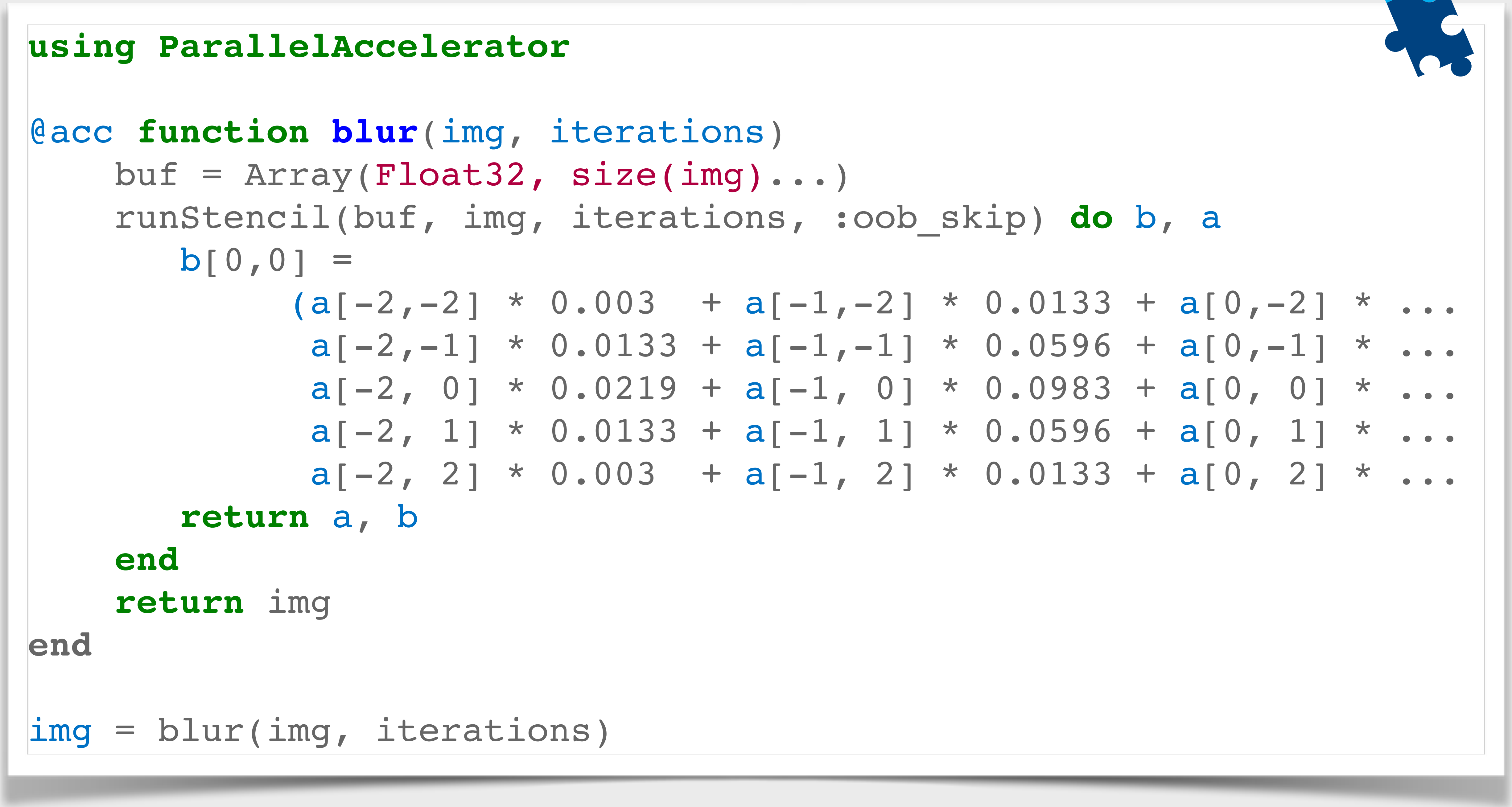
@acc example: Black-Scholes option pricing



Running on arrays of 100 million elements

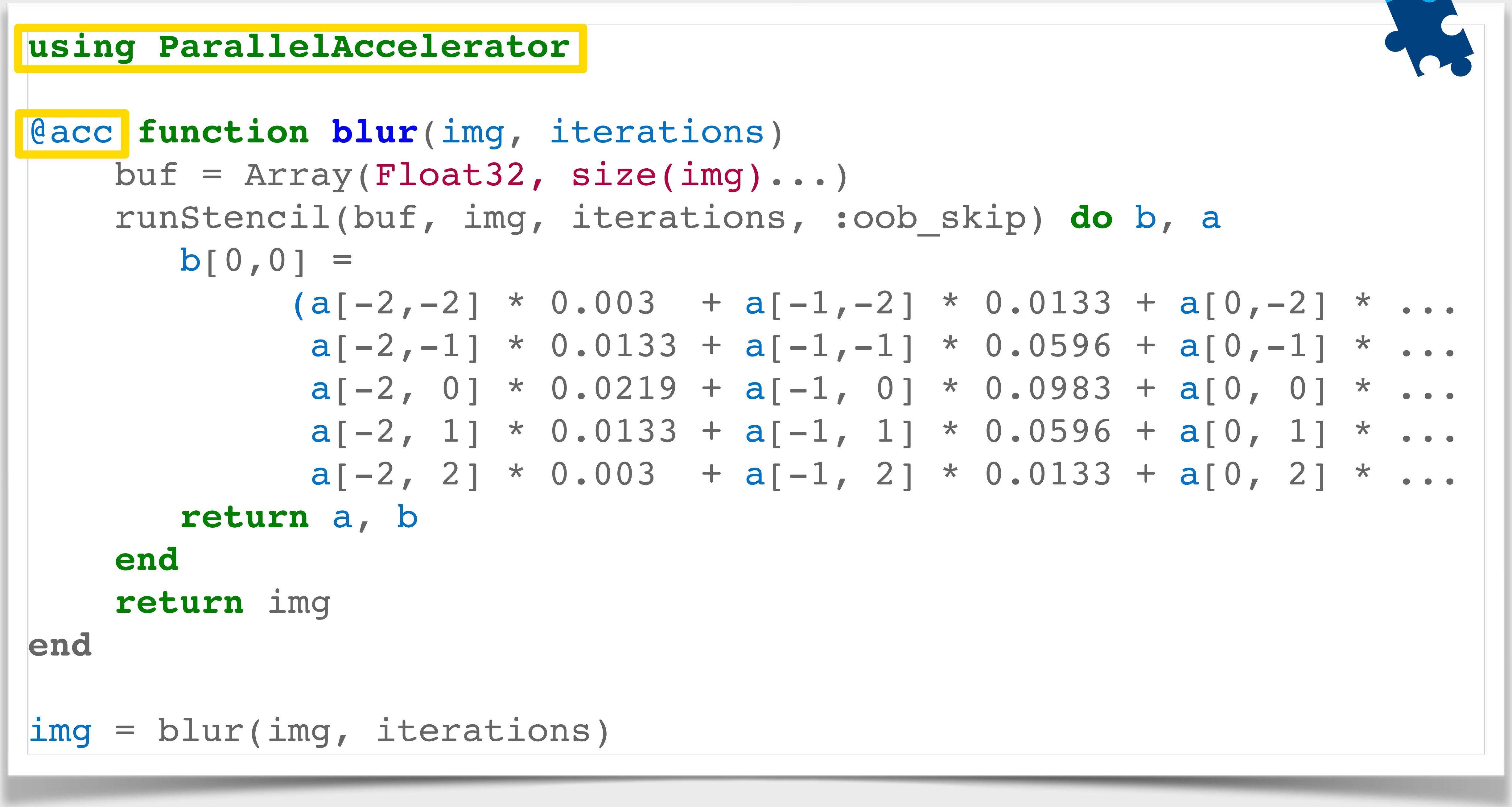
2 Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.3GHz processors, 18 cores each (36 cores total)128 GB RAM
Julia version 0.5.0; Matlab version R2015a

`runStencil` example: Gaussian blur



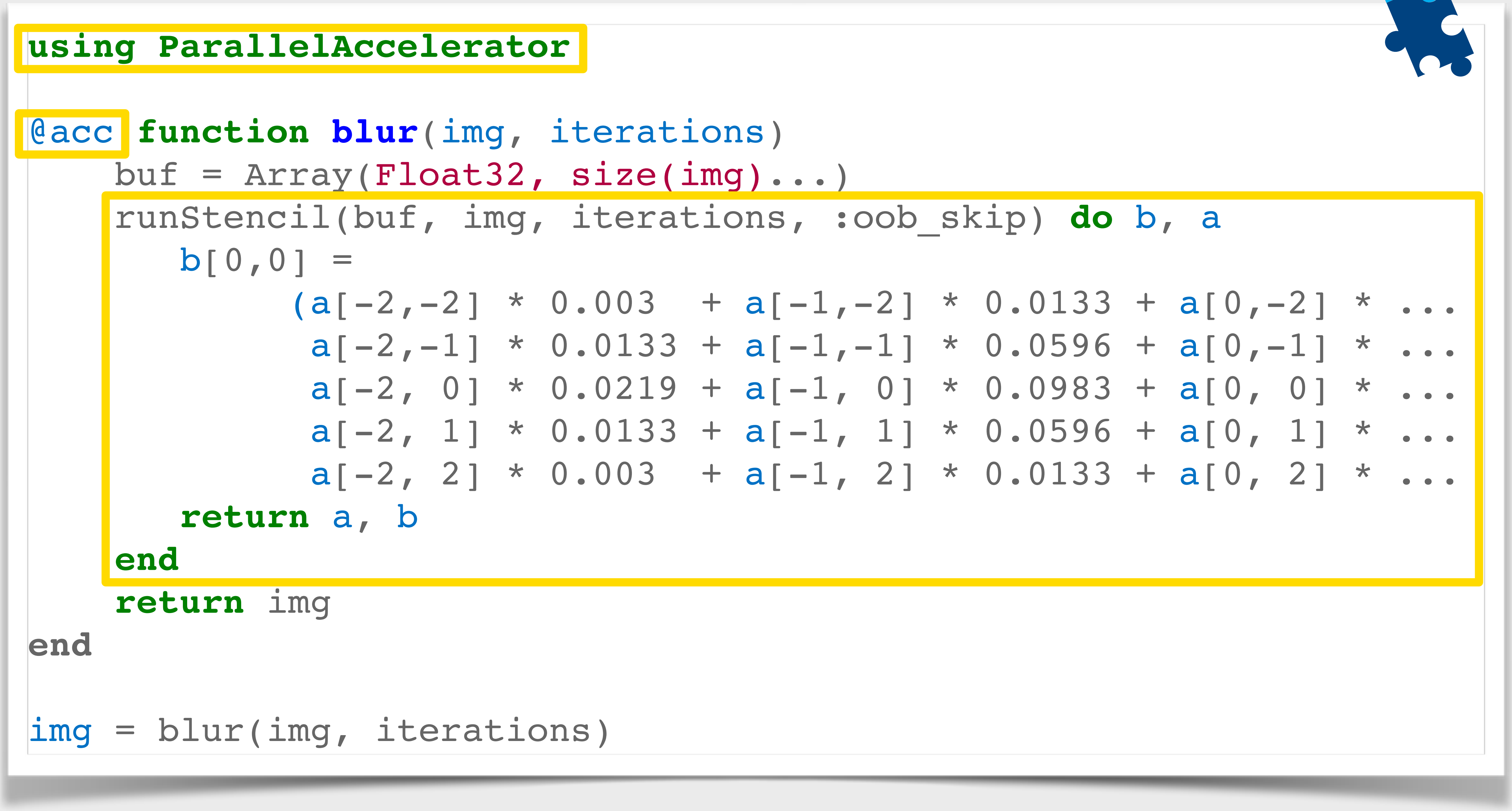
```
using ParallelAccelerator

@acc function blur(img, iterations)
    buf = Array(Float32, size(img)...)
    runStencil(buf, img, iterations, :oob_skip) do b, a
       b[0,0] =
            (a[-2,-2] * 0.003  + a[-1,-2] * 0.0133 + a[0,-2] * ...
             a[-2,-1] * 0.0133 + a[-1,-1] * 0.0596 + a[0,-1] * ...
             a[-2, 0] * 0.0219 + a[-1, 0] * 0.0983 + a[0, 0] * ...
             a[-2, 1] * 0.0133 + a[-1, 1] * 0.0596 + a[0, 1] * ...
             a[-2, 2] * 0.003  + a[-1, 2] * 0.0133 + a[0, 2] * ...
       return a, b
    end
    return img
end

img = blur(img, iterations)
```

runStencil example: Gaussian blur



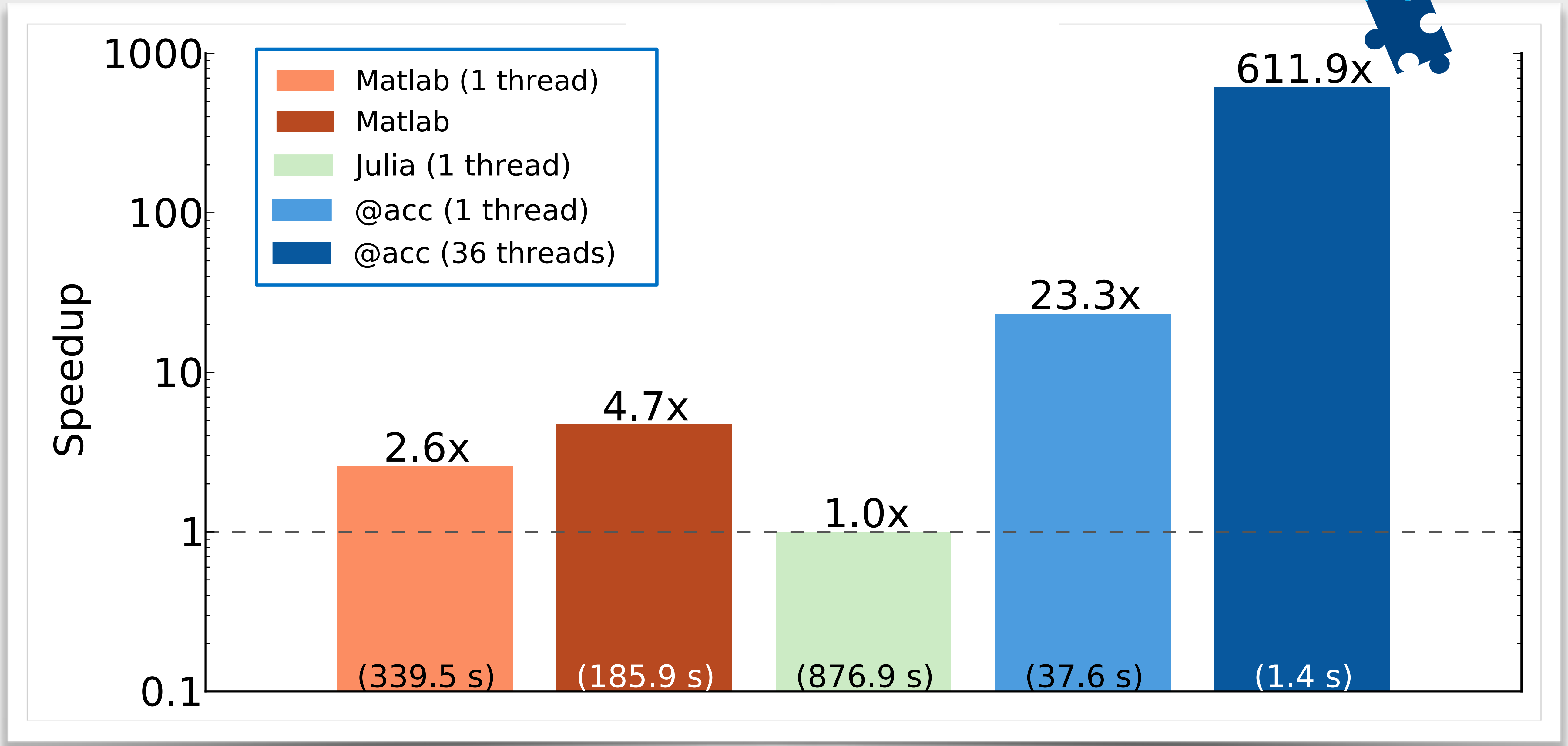
```
using ParallelAccelerator

@acc function blur(img, iterations)
    buf = Array(Float32, size(img)...)
    runStencil(buf, img, iterations, :oob_skip) do b, a
        b[0,0] =
                (a[-2,-2] * 0.003  + a[-1,-2] * 0.0133 + a[0,-2] * ...
                 a[-2,-1] * 0.0133 + a[-1,-1] * 0.0596 + a[0,-1] * ...
                 a[-2, 0] * 0.0219 + a[-1, 0] * 0.0983 + a[0, 0] * ...
                 a[-2, 1] * 0.0133 + a[-1, 1] * 0.0596 + a[0, 1] * ...
                 a[-2, 2] * 0.003  + a[-1, 2] * 0.0133 + a[0, 2] * ...
        return a, b
    end
    return img
end

img = blur(img, iterations)
```

runStencil example: Gaussian blur



```
using ParallelAccelerator

@acc function blur(img, iterations)
    buf = Array(Float32, size(img)...)
    runStencil(buf, img, iterations, :oob_skip) do b, a
       b[0,0] =
            (a[-2,-2] * 0.003  + a[-1,-2] * 0.0133 + a[0,-2] * ...
             a[-2,-1] * 0.0133 + a[-1,-1] * 0.0596 + a[0,-1] * ...
             a[-2, 0] * 0.0219 + a[-1, 0] * 0.0983 + a[0, 0] * ...
             a[-2, 1] * 0.0133 + a[-1, 1] * 0.0596 + a[0, 1] * ...
             a[-2, 2] * 0.003  + a[-1, 2] * 0.0133 + a[0, 2] * ...
       return a, b
    end
    return img
end

img = blur(img, iterations)
```

`runStencil` example: Gaussian blur

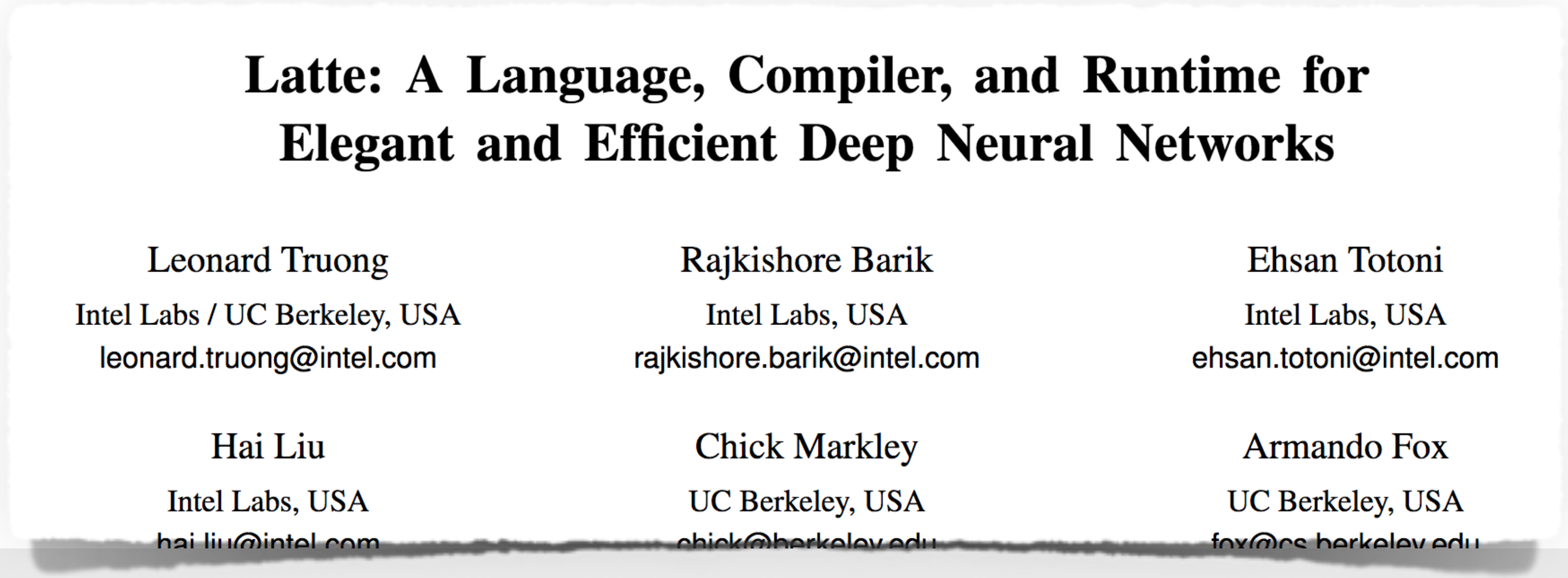


Running on a 7095x5322 source image for 100 iterations

2 Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.3GHz processors, 18 cores each (36 cores total) 128 GB RAM
Julia version 0.5.0; Matlab version R2015a

Impact of ParallelAccelerator...

[Truong *et al.*, PLDI '16]



Latte: A Language, Compiler, and Runtime for Elegant and Efficient Deep Neural Networks

Leonard Truong
Intel Labs / UC Berkeley, USA
leonard.truong@intel.com

Rajkishore Barik
Intel Labs, USA
rajkishore.barik@intel.com

Ehsan Totoni
Intel Labs, USA
ehsan.totoni@intel.com

Hai Liu
Intel Labs, USA
hai.liu@intel.com

Chick Markley
UC Berkeley, USA
chick@berkeley.edu

Armando Fox
UC Berkeley, USA
fox@cs.berkeley.edu

Impact of ParallelAccelerator...

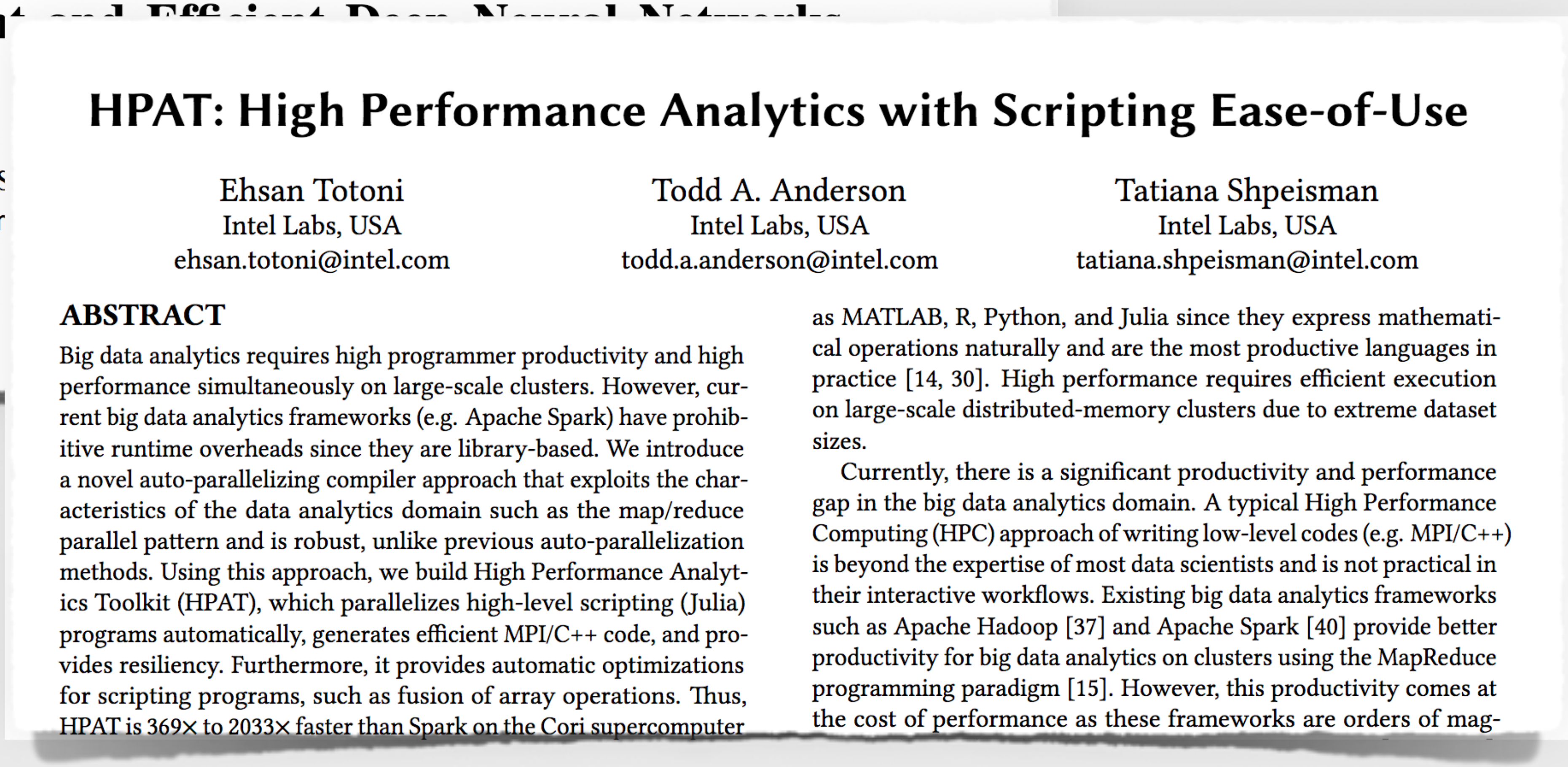
[Truong *et al.*, PLDI '16]

Latte: A Language, Compiler, and Runtime for Elegant and Efficient Deep Neural Networks

Leonard Truong
Intel Labs / UC Berkeley, US
leonard.truong@intel.com

Hai Liu
Intel Labs, USA
hai.liu@intel.com

[Totoni *et al.*, ICS '17]



HPAT: High Performance Analytics with Scripting Ease-of-Use

Ehsan Totoni
Intel Labs, USA
ehsan.totoni@intel.com

Todd A. Anderson
Intel Labs, USA
todd.a.anderson@intel.com

Tatiana Shpeisman
Intel Labs, USA
tatiana.shpeisman@intel.com

ABSTRACT

Big data analytics requires high programmer productivity and high performance simultaneously on large-scale clusters. However, current big data analytics frameworks (e.g. Apache Spark) have prohibitive runtime overheads since they are library-based. We introduce a novel auto-parallelizing compiler approach that exploits the characteristics of the data analytics domain such as the map/reduce parallel pattern and is robust, unlike previous auto-parallelization methods. Using this approach, we build High Performance Analytics Toolkit (HPAT), which parallelizes high-level scripting (Julia) programs automatically, generates efficient MPI/C++ code, and provides resiliency. Furthermore, it provides automatic optimizations for scripting programs, such as fusion of array operations. Thus, HPAT is 369× to 2033× faster than Spark on the Cori supercomputer

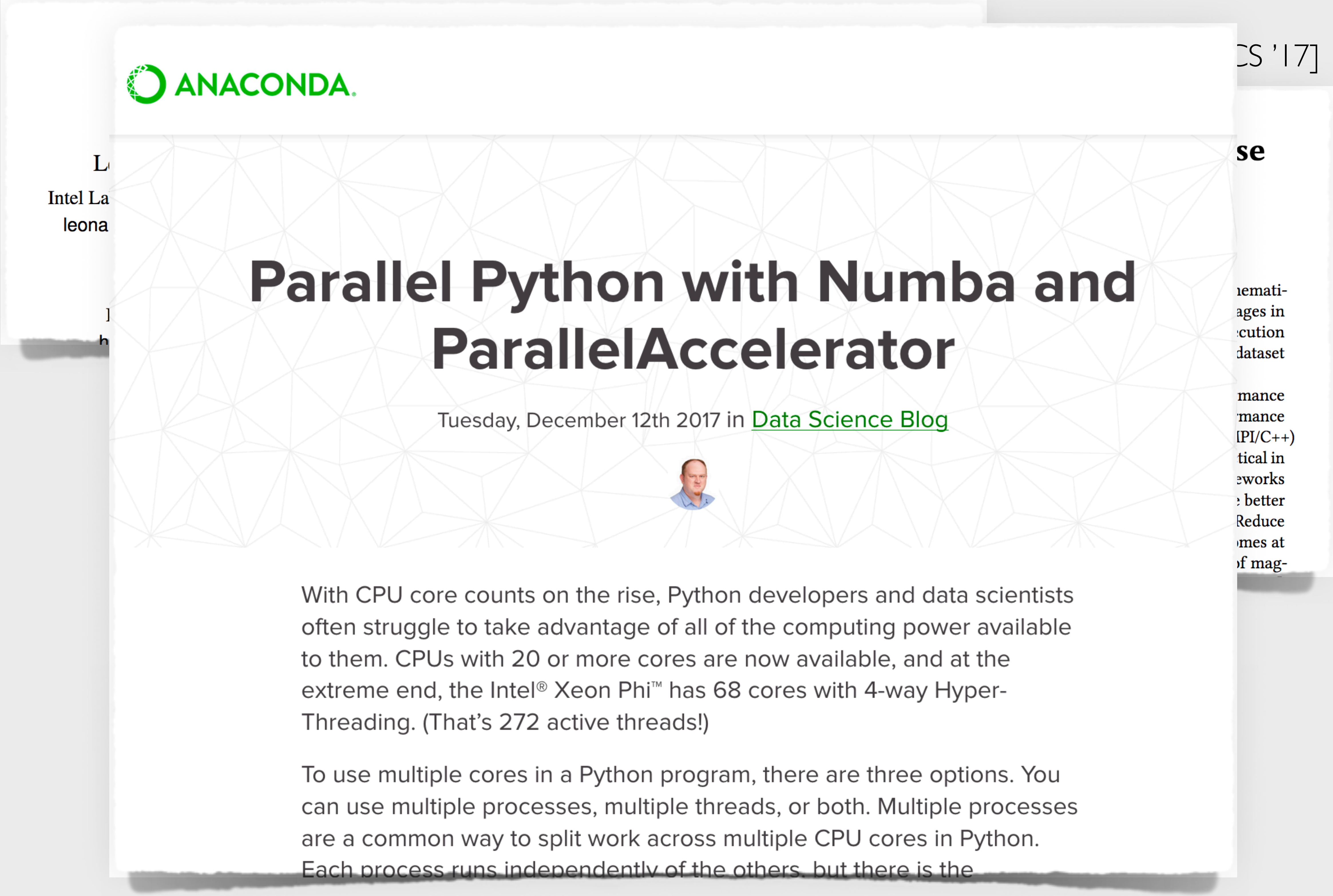
as MATLAB, R, Python, and Julia since they express mathematical operations naturally and are the most productive languages in practice [14, 30]. High performance requires efficient execution on large-scale distributed-memory clusters due to extreme dataset sizes.

Currently, there is a significant productivity and performance gap in the big data analytics domain. A typical High Performance Computing (HPC) approach of writing low-level codes (e.g. MPI/C++) is beyond the expertise of most data scientists and is not practical in their interactive workflows. Existing big data analytics frameworks such as Apache Hadoop [37] and Apache Spark [40] provide better productivity for big data analytics on clusters using the MapReduce programming paradigm [15]. However, this productivity comes at the cost of performance as these frameworks are orders of mag-

Impact of ParallelAccelerator...

[Truong *et al.*, PLDI '16]

CS '17]



L

Intel La

leona

]

h

se

nemati-
ages in
cution
dataset

mance
rmance
IPI/C++)
tical in
eworks
better
Reduce
mes at
of mag-

ANACONDA.

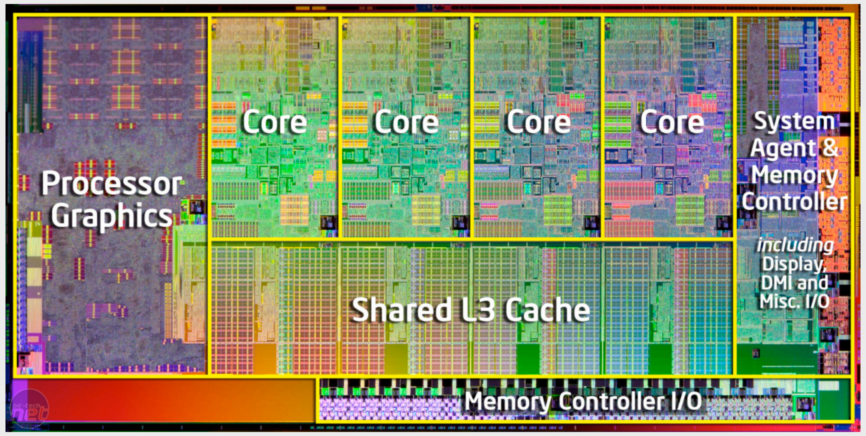
Parallel Python with Numba and ParallelAccelerator

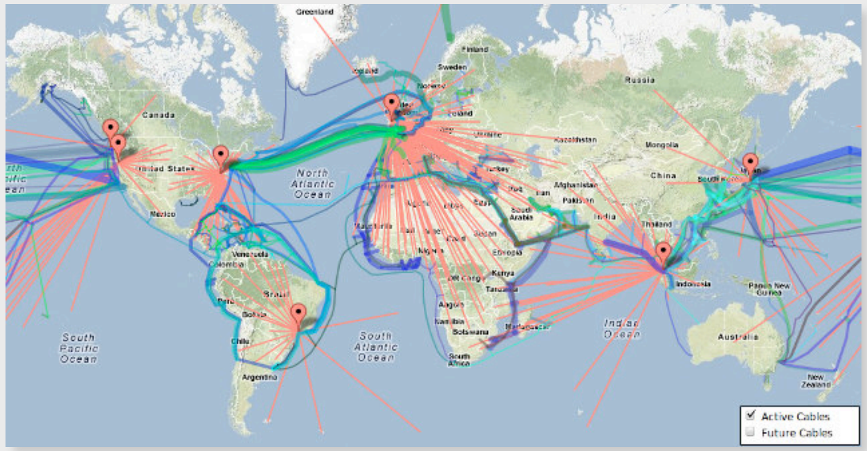
Tuesday, December 12th 2017 in Data Science Blog

With CPU core counts on the rise, Python developers and data scientists often struggle to take advantage of all of the computing power available to them. CPUs with 20 or more cores are now available, and at the extreme end, the Intel® Xeon Phi™ has 68 cores with 4-way Hyper-Threading. (That's 272 active threads!)

To use multiple cores in a Python program, there are three options. You can use multiple processes, multiple threads, or both. Multiple processes are a common way to split work across multiple CPU cores in Python. Each process runs independently of the others, but there is the

Source: www.anaconda.com/blog/developer-blog/parallel-python-with-numba-and-parallelaccelerator/

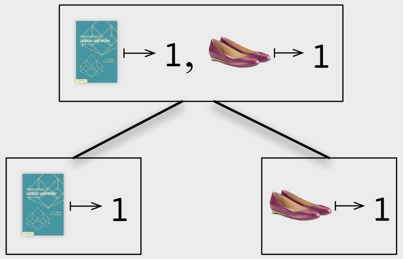




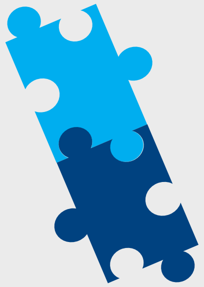
Parallel systems

Distributed systems

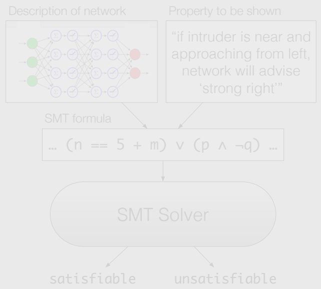
language-based approaches



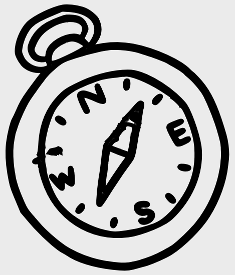
Lattice-based data structures (LVars)
for deterministic programming
[POPL '14, PLDI '14, FHPC '13, WoDet '14]



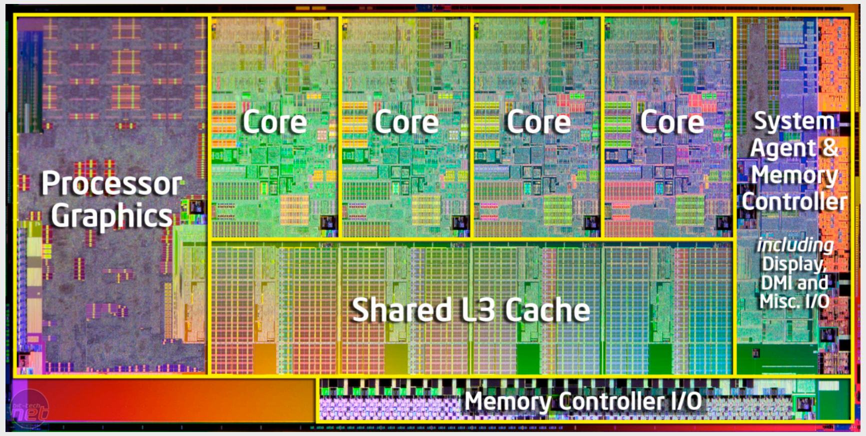
Non-invasive DSLs for
productive parallelism
[ECOOP '17]

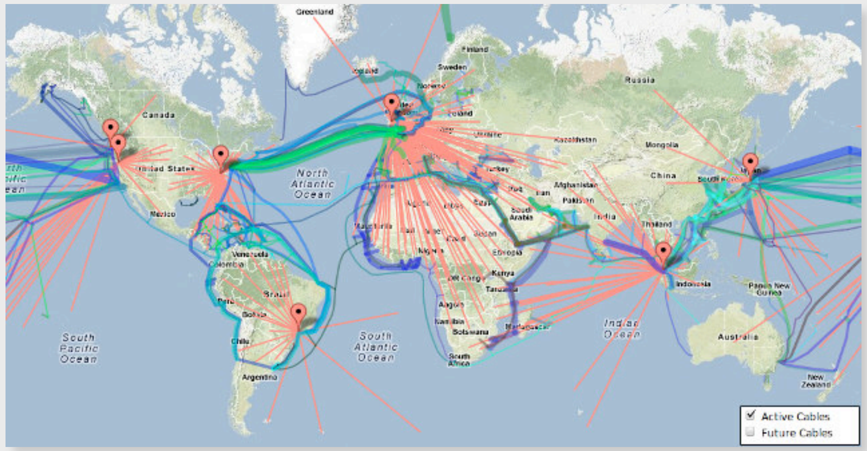

Description of network
Property to be shown
"if intruder is near and approaching from left, network will advise strong right"
SMT formula
... (n == 5 + m) ∨ (p ∧ ¬q) ...
SMT Solver
satisfiable
unsatisfiable

SMT-based verification
of neural networks
[SysML '18]



Guiding principle:
Find the right **high-level abstractions**
to enable **efficient** computation

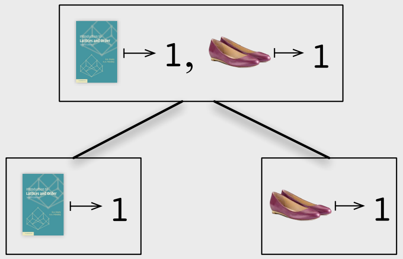




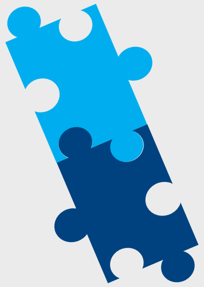
Parallel systems

Distributed systems

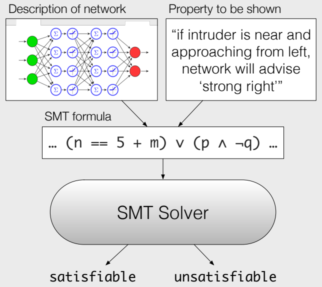
language-based approaches



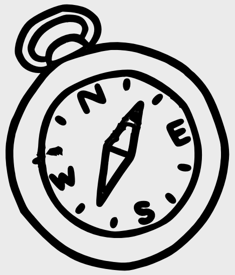
Lattice-based data structures (LVars) for deterministic programming
[POPL '14, PLDI '14, FHPC '13, WoDet '14]



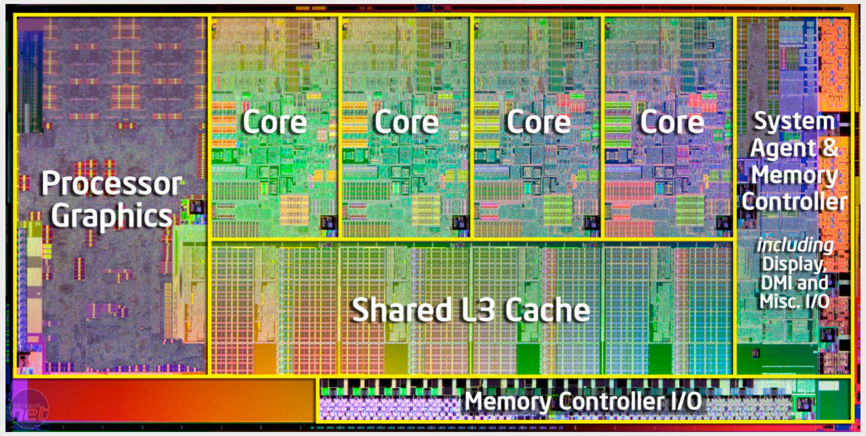
Non-invasive DSLs for productive parallelism
[ECOOP '17]

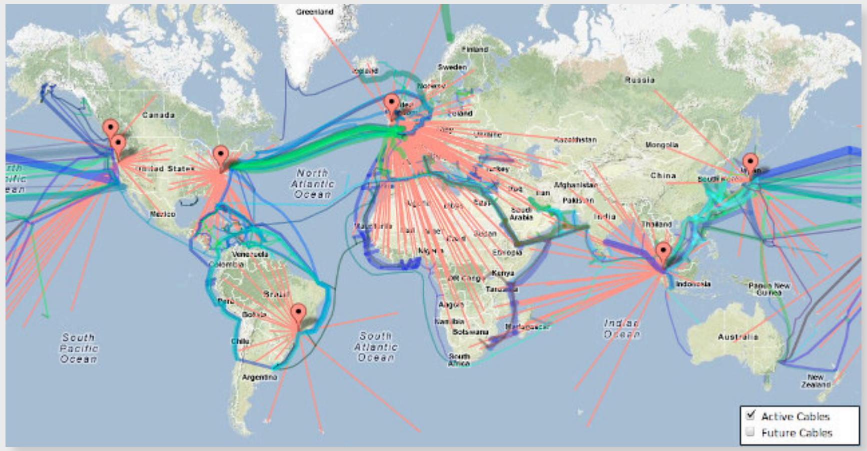

Description of network
Property to be shown
"if intruder is near and approaching from left, network will advise 'strong right'"
SMT formula
… (n == 5 + m) ∨ (p ∧ ¬q) …
SMT Solver
satisfiable
unsatisfiable

SMT-based verification of neural networks
[SysML '18]



Guiding principle:
Find the right **high-level abstractions** to enable **efficient** computation

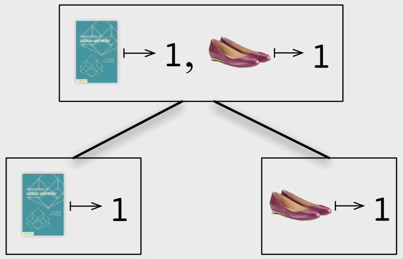


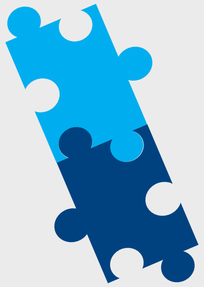


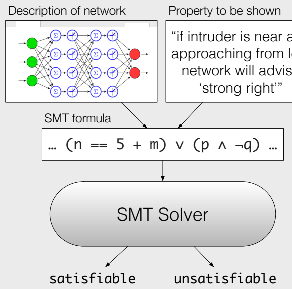
Parallel systems

Distributed systems

language-based approaches






Description of network

Property to be shown

"if intruder is near and approaching from left, network will advise 'strong right'"

SMT formula

… (n == 5 + m) ∨ (p ∧ ¬q) …

SMT Solver

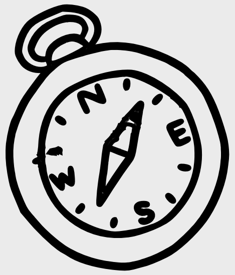
satisfiable

unsatisfiable

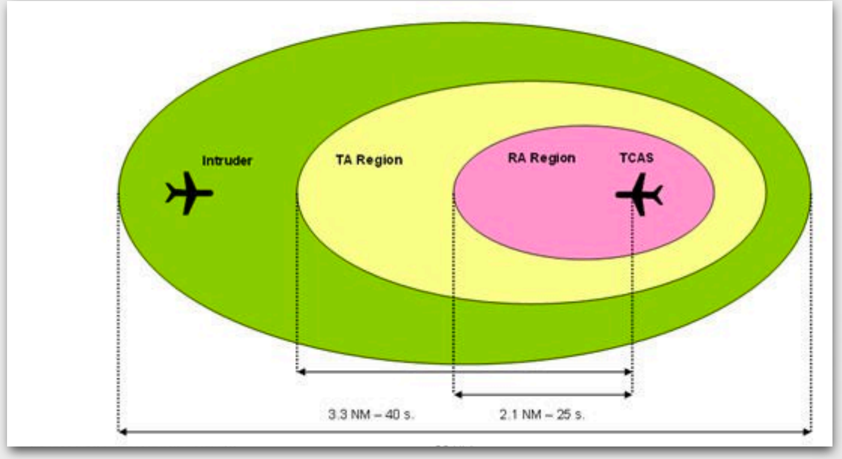
Lattice-based data structures (LVars) for deterministic programming
[POPL '14, PLDI '14, FHPC '13, WoDet '14]

Non-invasive DSLs for productive parallelism
[ECOOP '17]

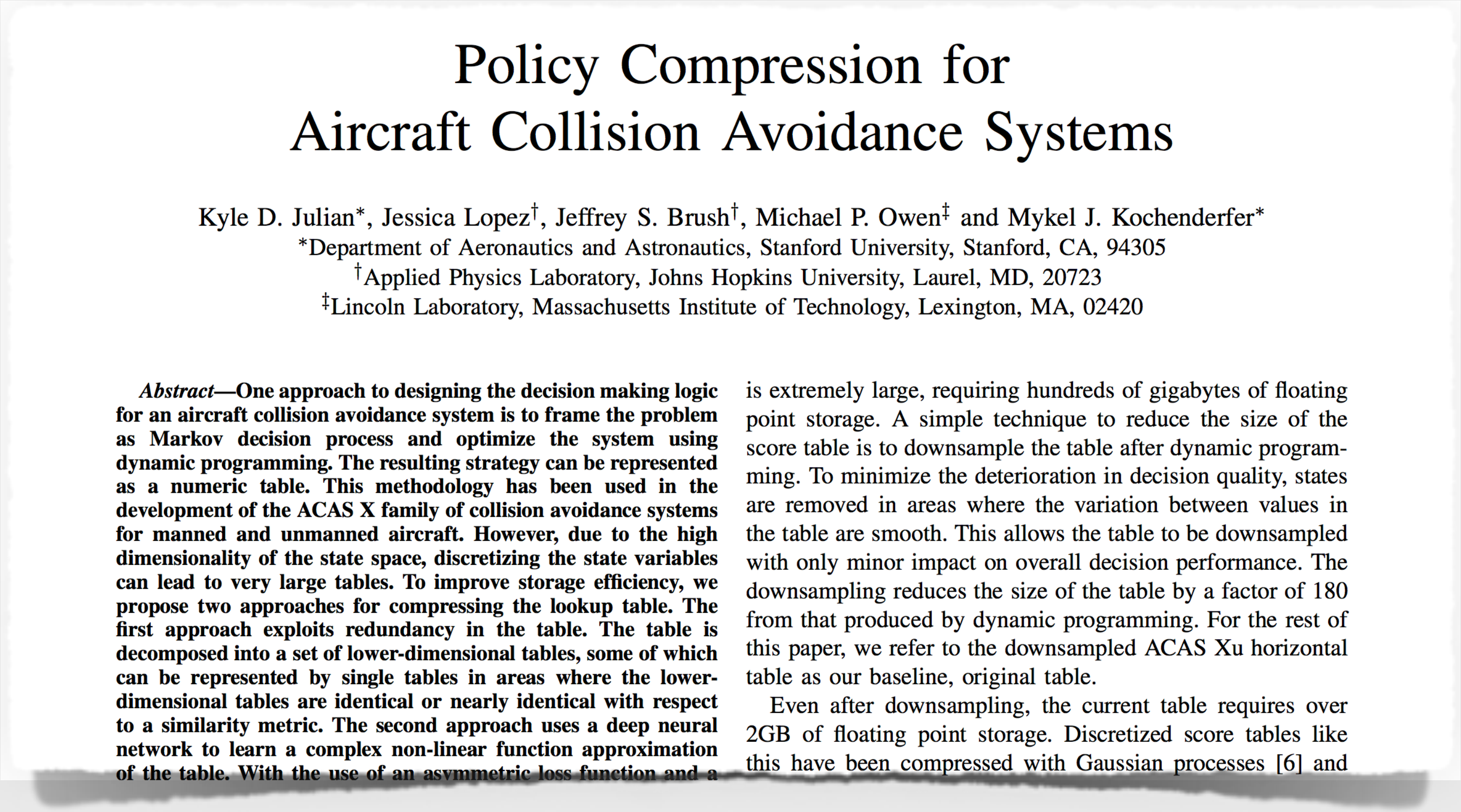
SMT-based verification of neural networks
[SysML '18]



Guiding principle:
Find the right **high-level abstractions**
to enable **efficient** computation



[Julian *et al.*, 2016]



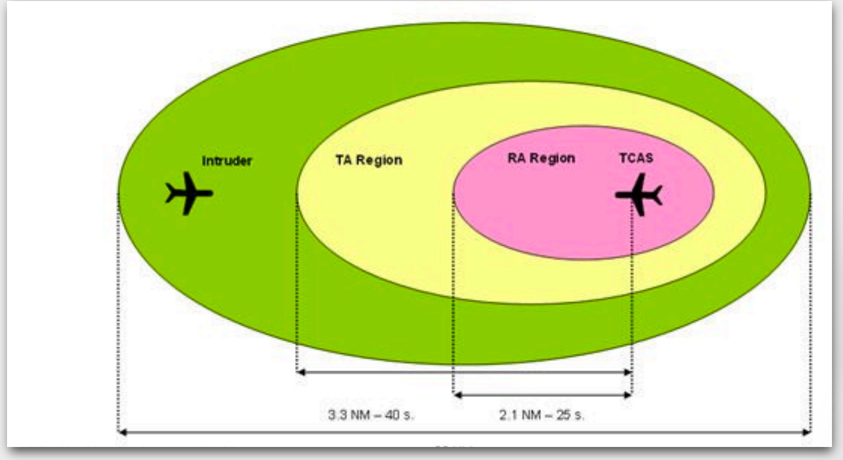
Policy Compression for Aircraft Collision Avoidance Systems

Kyle D. Julian*, Jessica Lopez[†], Jeffrey S. Brush[†], Michael P. Owen[‡] and Mykel J. Kochenderfer*
*Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, 94305
[†]Applied Physics Laboratory, Johns Hopkins University, Laurel, MD, 20723
[‡]Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, MA, 02420

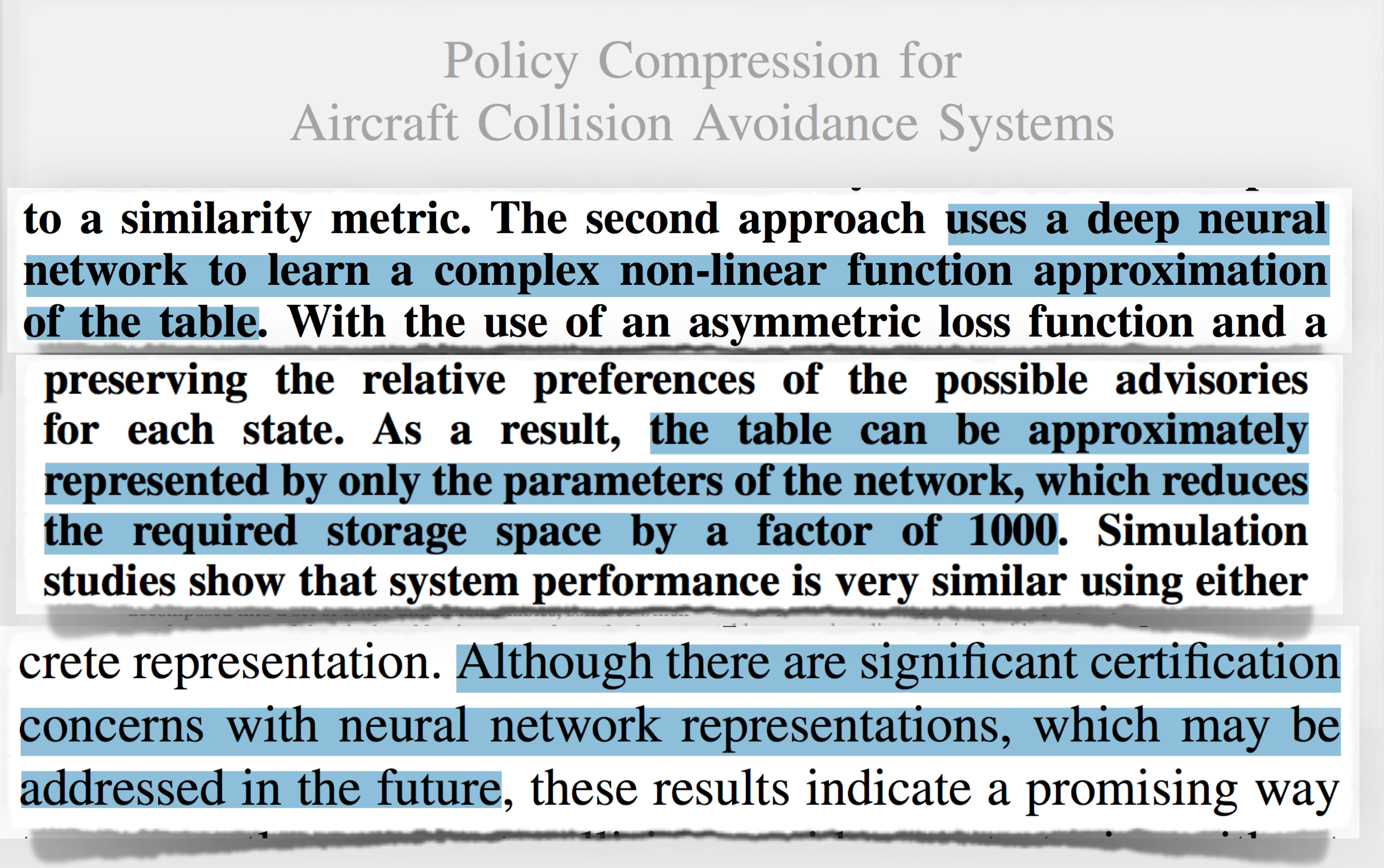
***Abstract*—One approach to designing the decision making logic for an aircraft collision avoidance system is to frame the problem as Markov decision process and optimize the system using dynamic programming. The resulting strategy can be represented as a numeric table. This methodology has been used in the development of the ACAS X family of collision avoidance systems for manned and unmanned aircraft. However, due to the high dimensionality of the state space, discretizing the state variables can lead to very large tables. To improve storage efficiency, we propose two approaches for compressing the lookup table. The first approach exploits redundancy in the table. The table is decomposed into a set of lower-dimensional tables, some of which can be represented by single tables in areas where the lower-dimensional tables are identical or nearly identical with respect to a similarity metric. The second approach uses a deep neural network to learn a complex non-linear function approximation of the table. With the use of an asymmetric loss function and a**

is extremely large, requiring hundreds of gigabytes of floating point storage. A simple technique to reduce the size of the score table is to downsample the table after dynamic programming. To minimize the deterioration in decision quality, states are removed in areas where the variation between values in the table are smooth. This allows the table to be downsampled with only minor impact on overall decision performance. The downsampling reduces the size of the table by a factor of 180 from that produced by dynamic programming. For the rest of this paper, we refer to the downsampled ACAS Xu horizontal table as our baseline, original table.

Even after downsampling, the current table requires over 2GB of floating point storage. Discretized score tables like this have been compressed with Gaussian processes [6] and



[Julian *et al.*, 2016]



Policy Compression for
Aircraft Collision Avoidance Systems

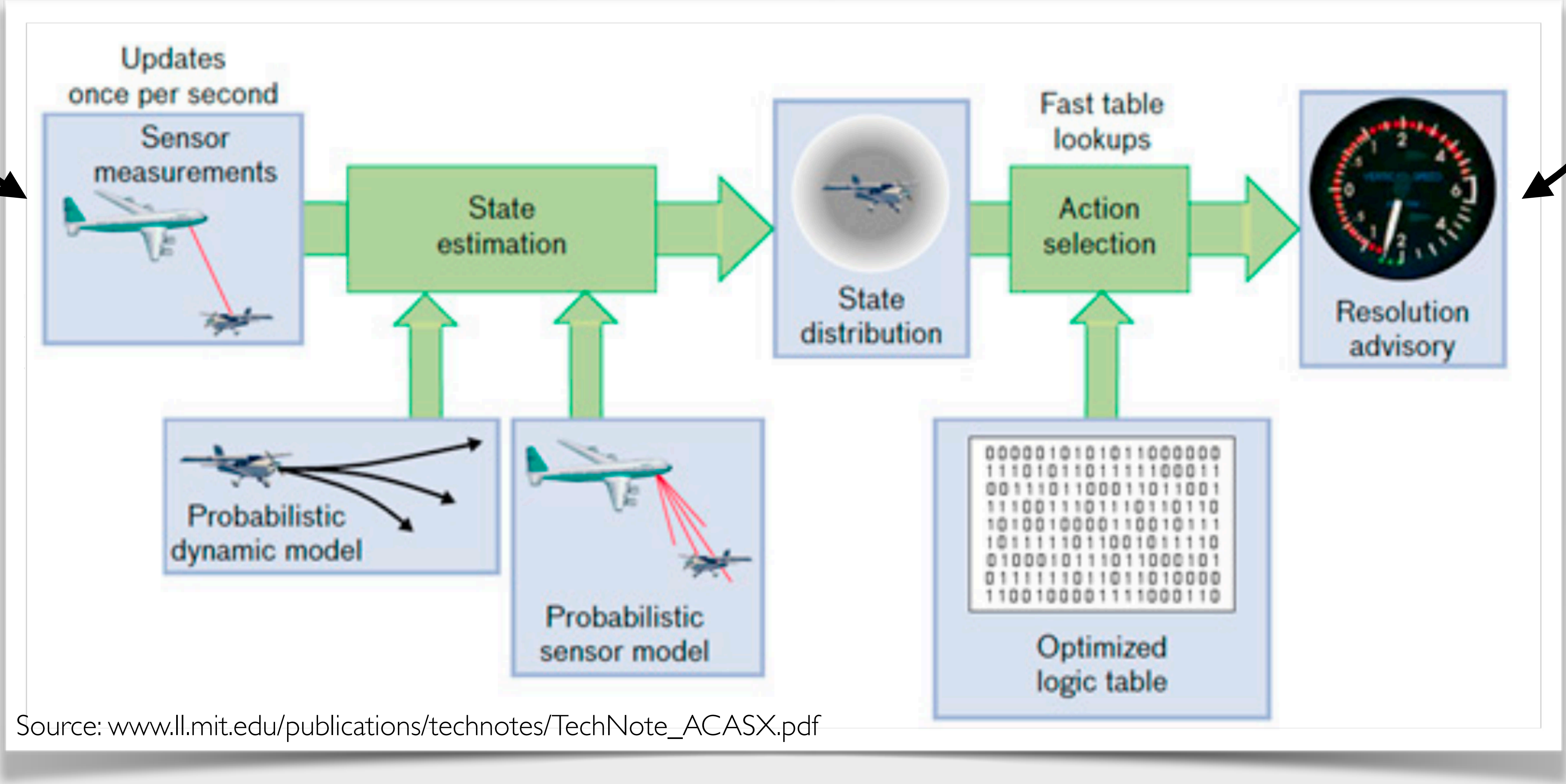
to a similarity metric. The second approach uses a deep neural network to learn a complex non-linear function approximation of the table. With the use of an asymmetric loss function and a

preserving the relative preferences of the possible advisories for each state. As a result, the table can be approximately represented by only the parameters of the network, which reduces the required storage space by a factor of 1000. Simulation studies show that system performance is very similar using either

crete representation. Although there are significant certification concerns with neural network representations, which may be addressed in the future, these results indicate a promising way

Input: sensor data once per second

Output: one of five resolution advisories (COC, weak left, weak right, strong left, strong right)



Source: www.ll.mit.edu/publications/technotes/TechNote_ACASX.pdf

Ir

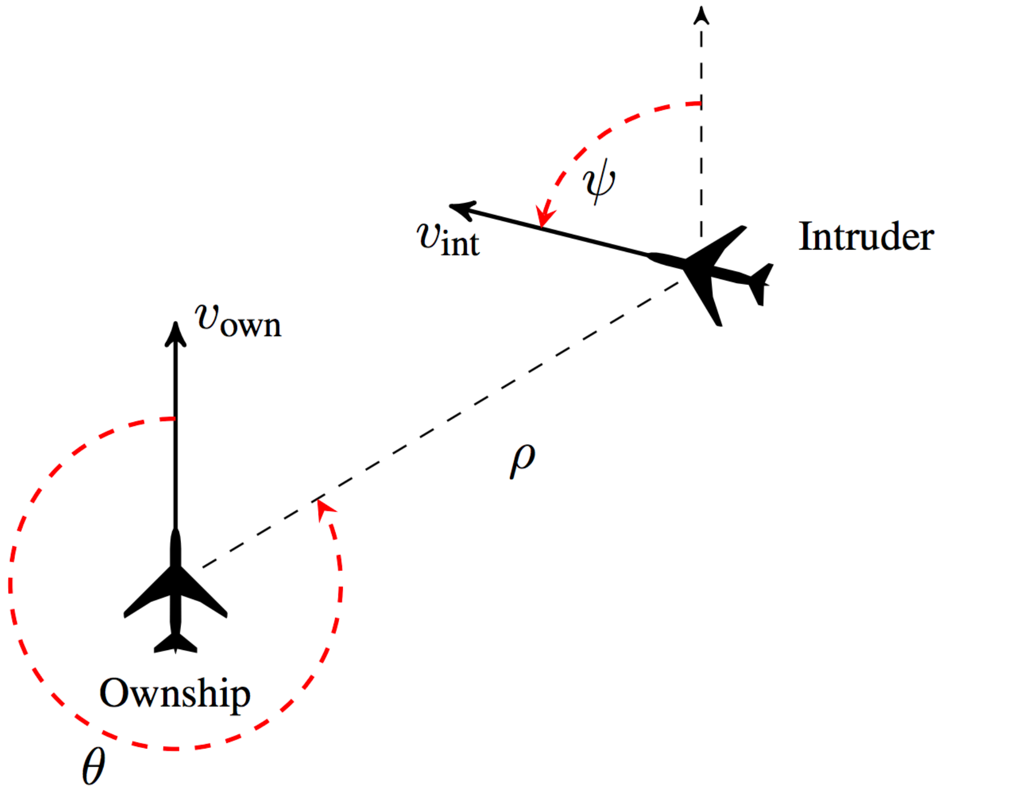
c

right)

[Julian *et al.*, 2016]

~120M 7-dimensional states

- ρ: distance from ownship to intruder
- θ: angle to intruder
- ψ: heading angle of intruder
- v_{own}: speed of ownship
- v_{int}: speed of intruder
- τ: time until loss of vertical separation
- a_{prev}: previous advisory



Probabilistic dynamic model
Probabilistic sensor model
Optimized logic table

Source: www.ll.mit.edu/publications/technotes/TechNote_ACASX.pdf

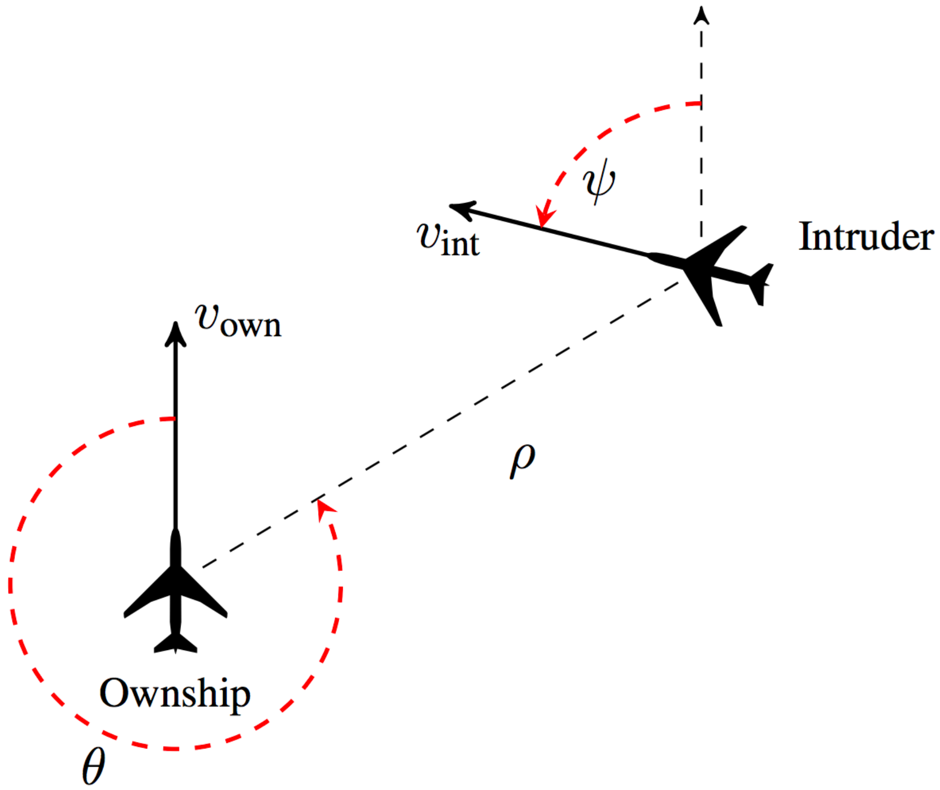
Ir

c right)

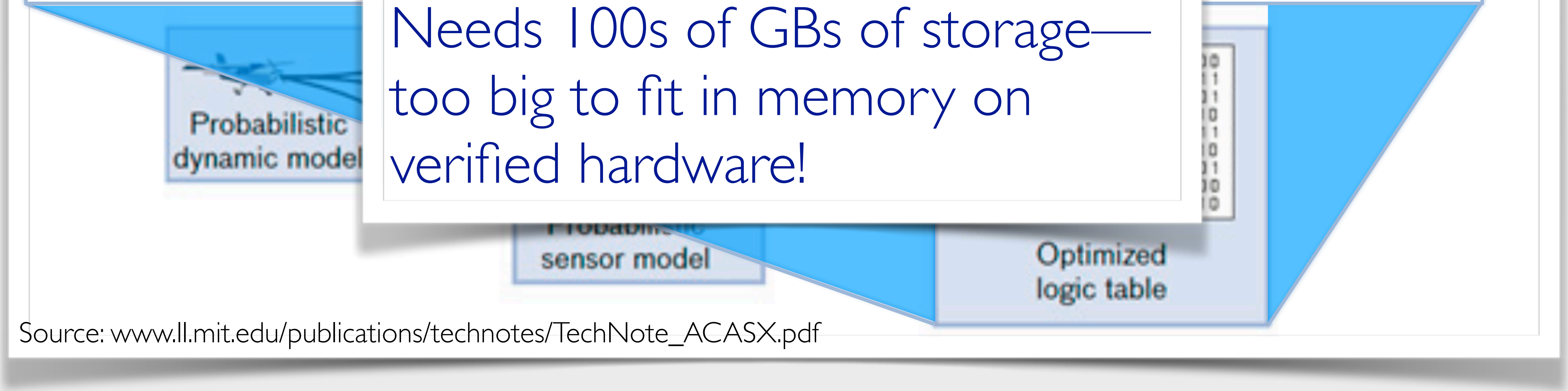
[Julian *et al.*, 2016]

~120M 7-dimensional states

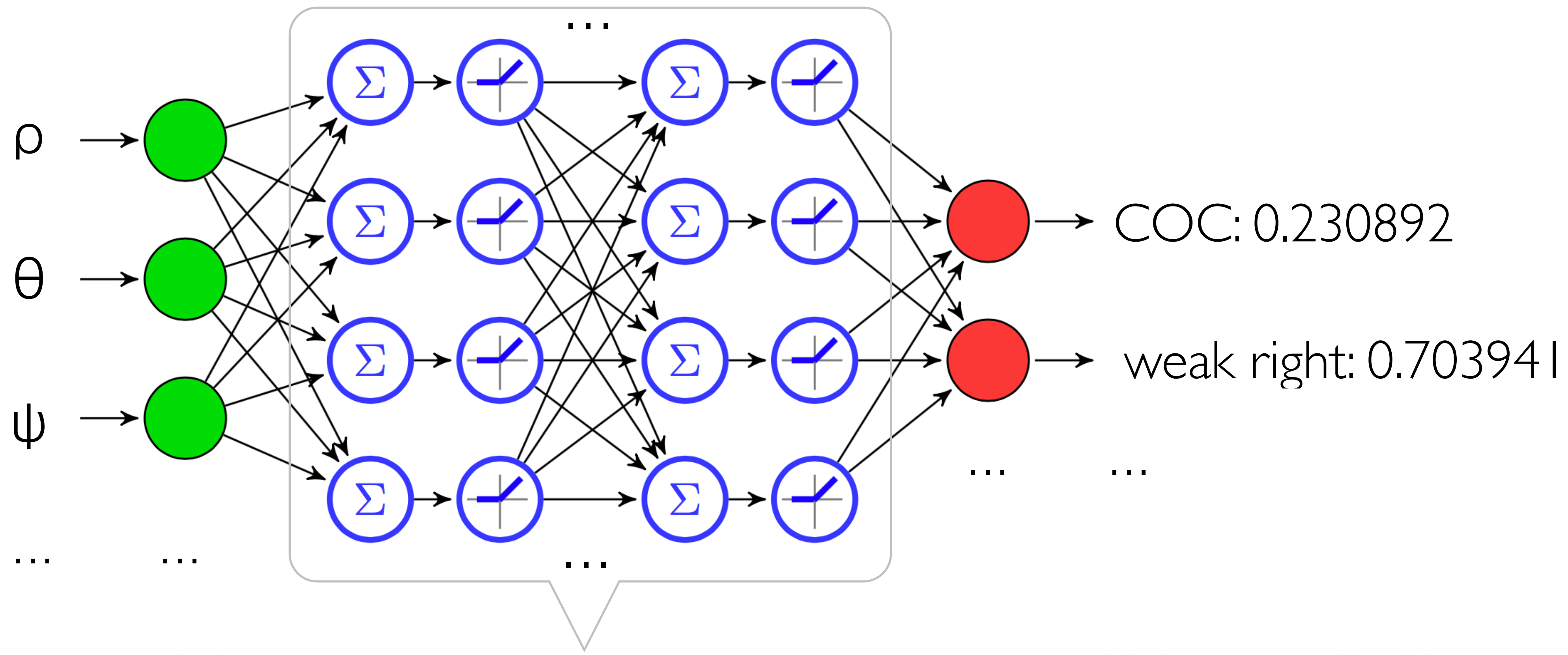
- ρ: distance from ownship to intruder
- θ: angle to intruder
- ψ: heading angle of intruder
- v_{own}: speed of ownship
- v_{int}: speed of intruder
- τ: time until loss of vertical separation
- a_{prev}: previous advisory



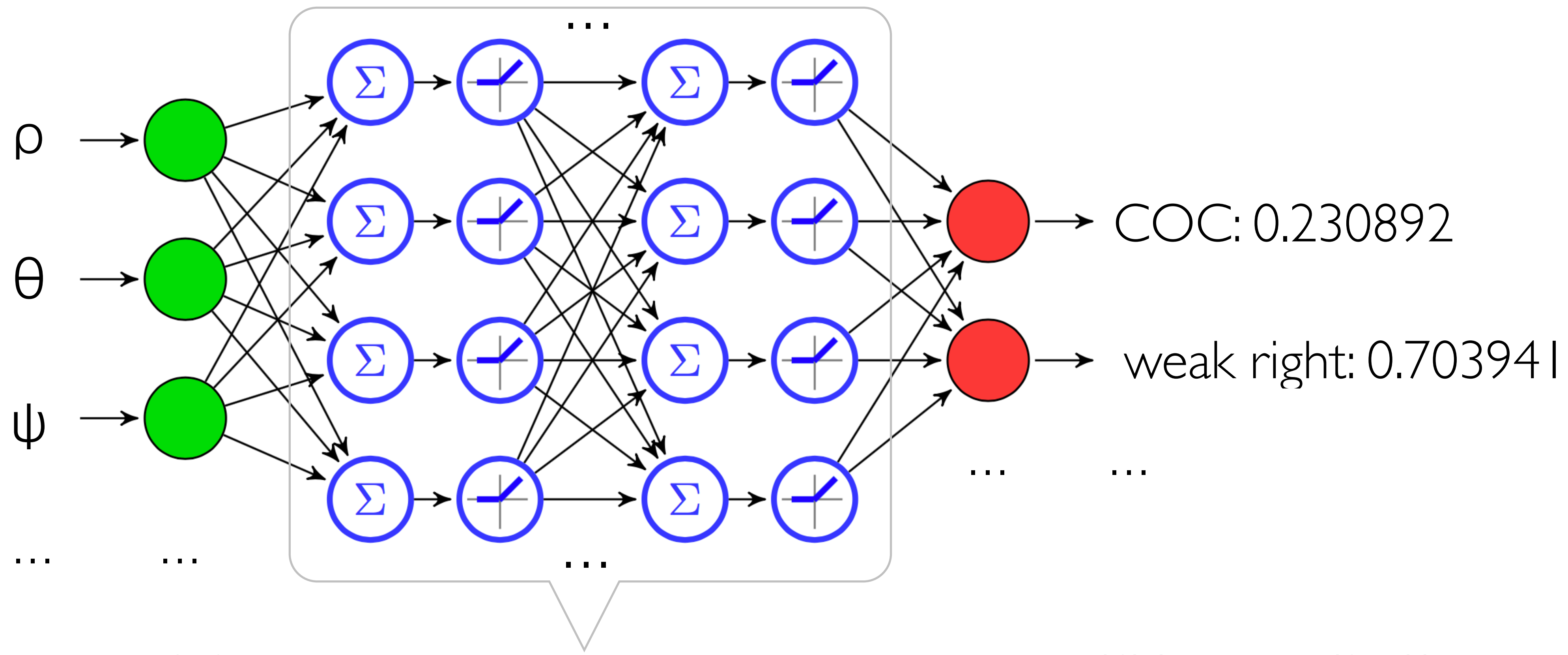
Needs 100s of GBs of storage—too big to fit in memory on verified hardware!



Source: www.ll.mit.edu/publications/technotes/TechNote_ACASX.pdf



9 fully-connected layers with ReLU activations ($f(x) = \max(0, x)$)

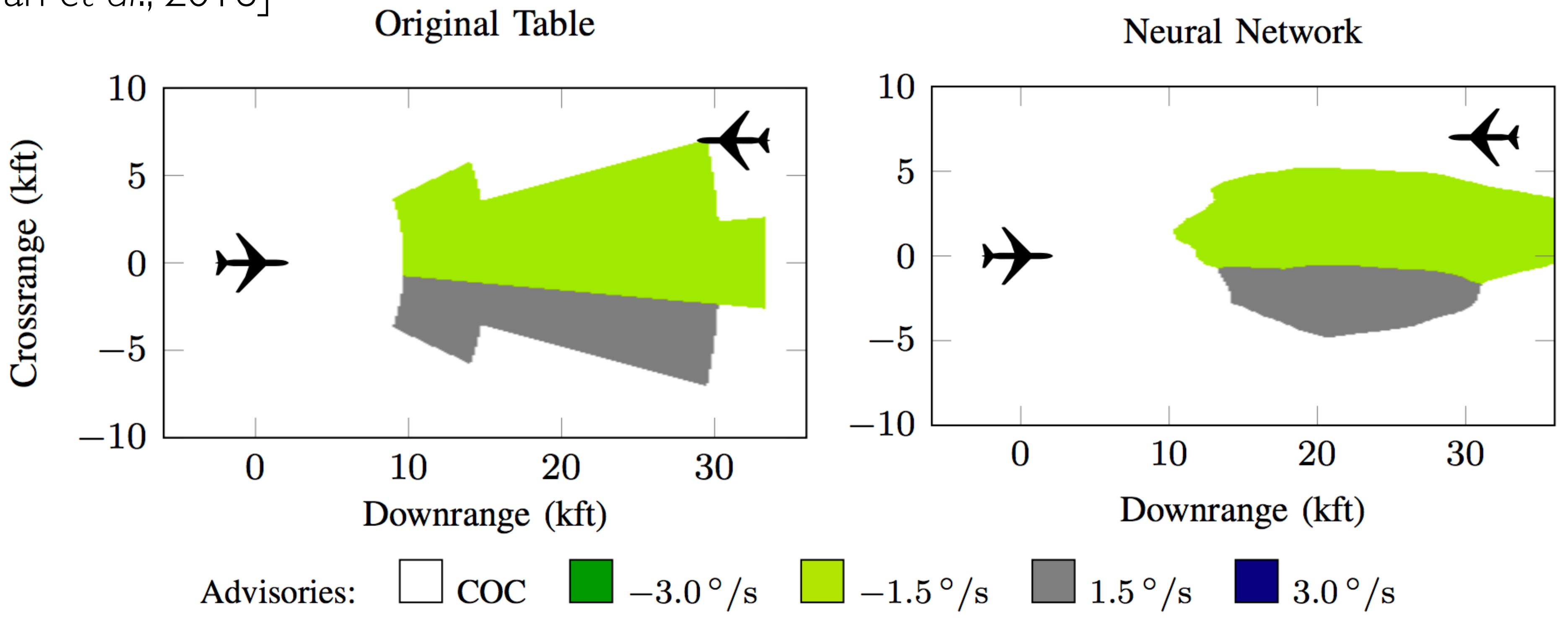


COC: 0.230892

weak right: 0.703941

9 fully-connected layers with ReLU activations ($f(x) = \max(0, x)$)

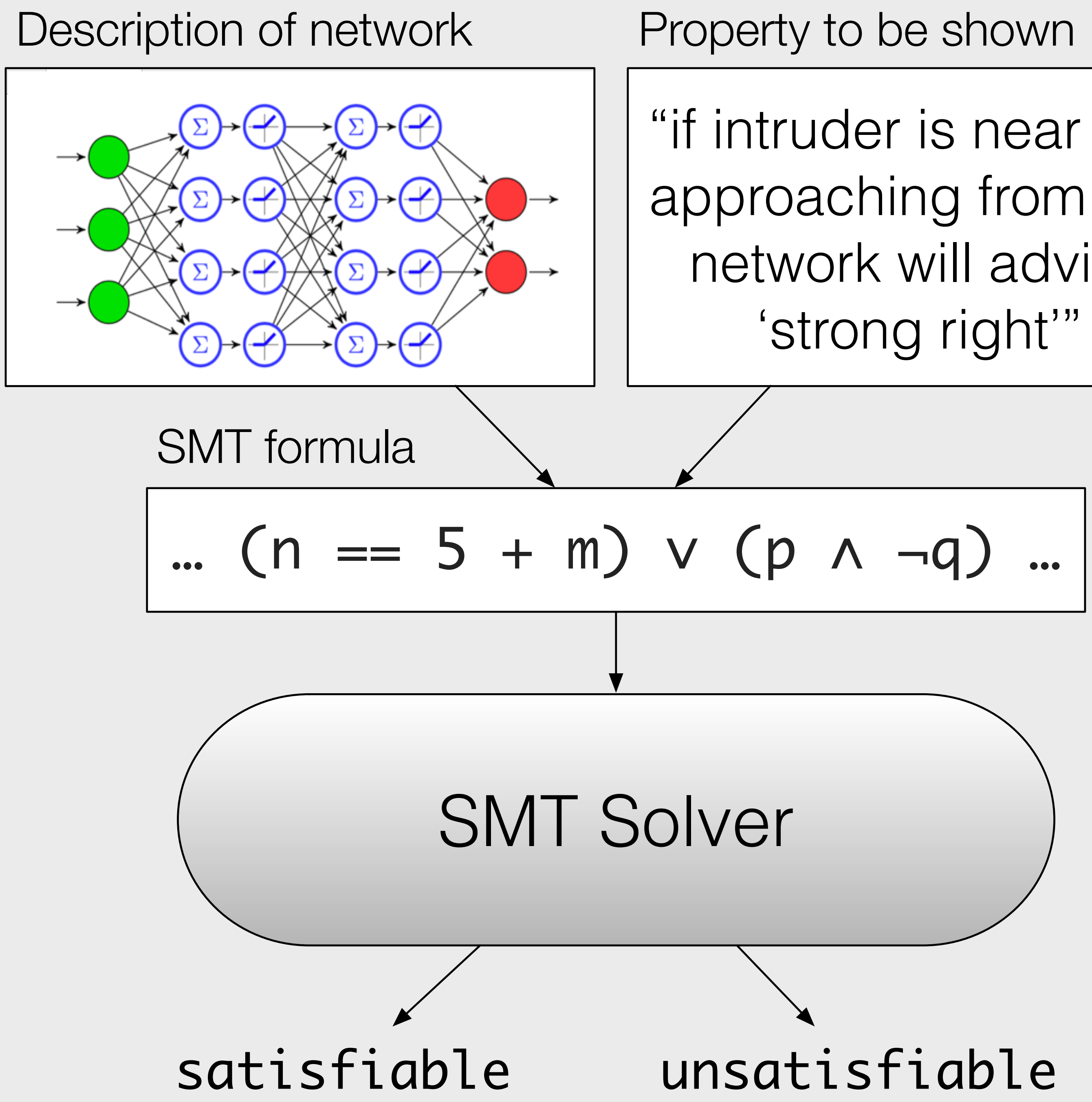
[Julian *et al.*, 2016]



Verification strategy

SAT: determine if a Boolean formula (containing only Boolean variables, parens, ∧, ∨, ¬) is satisfiable

SMT: determine satisfiability of a formula with respect to some *theory* (e.g., theory of linear real arithmetic)



Description of network

Property to be shown

"if intruder is near and approaching from left, network will advise 'strong right'"

SMT formula

```
… (n == 5 + m) ∨ (p ∧ ¬q) …
```

SMT Solver

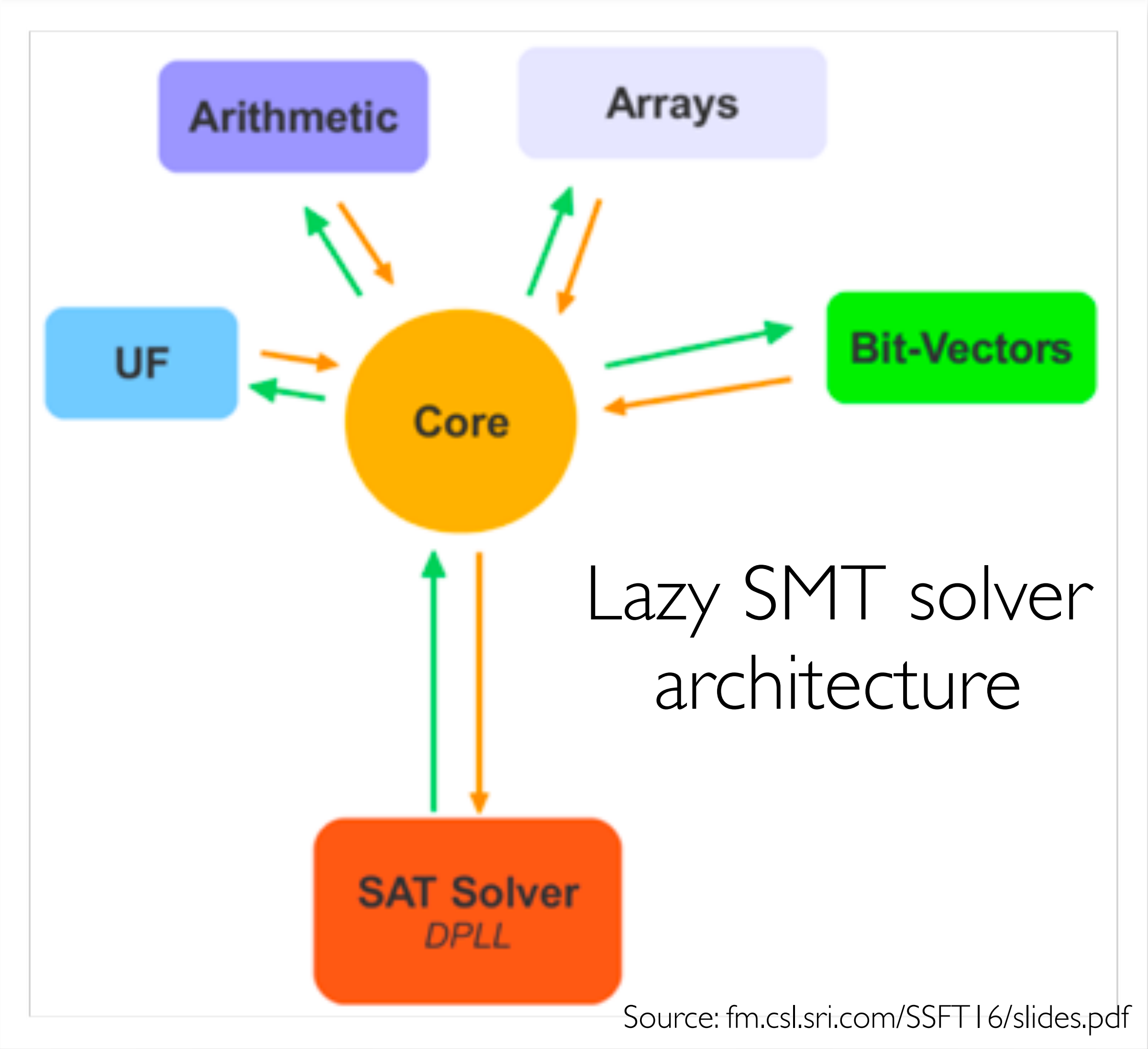
`satisfiable`

`unsatisfiable`

The virtues of laziness

Eager approach: convert whole SMT formula to SAT formula immediately, then solve with SAT solver

Lazy approach: use *theory solvers*, each specific to a particular theory



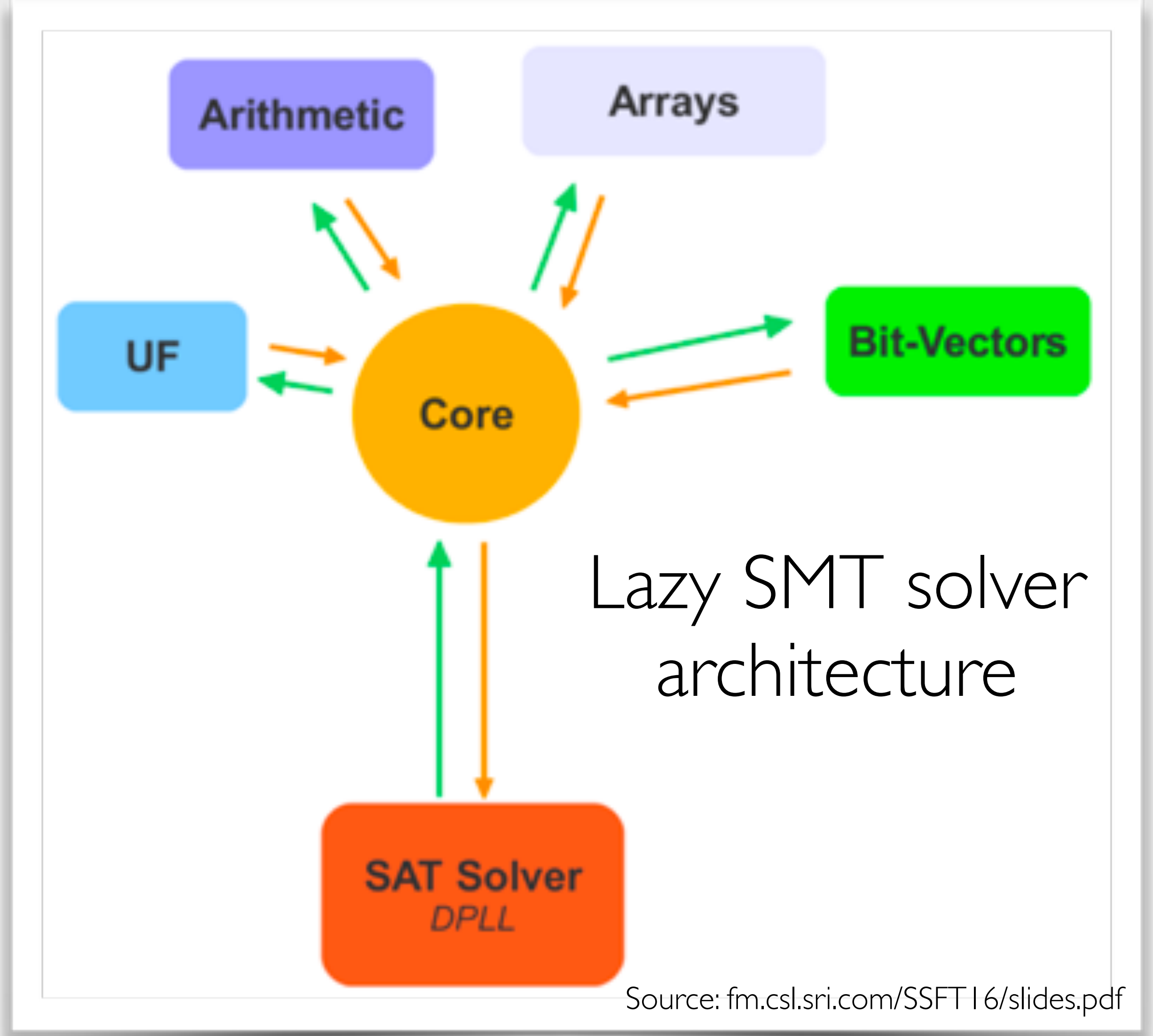
Lazy SMT solver architecture

Source: fm.csl.sri.com/SSFT16/slides.pdf

The virtues of laziness

Eager approach: convert whole SMT formula to SAT formula immediately, then solve with SAT solver

Lazy approach: use *theory solvers*, each specific to a particular theory



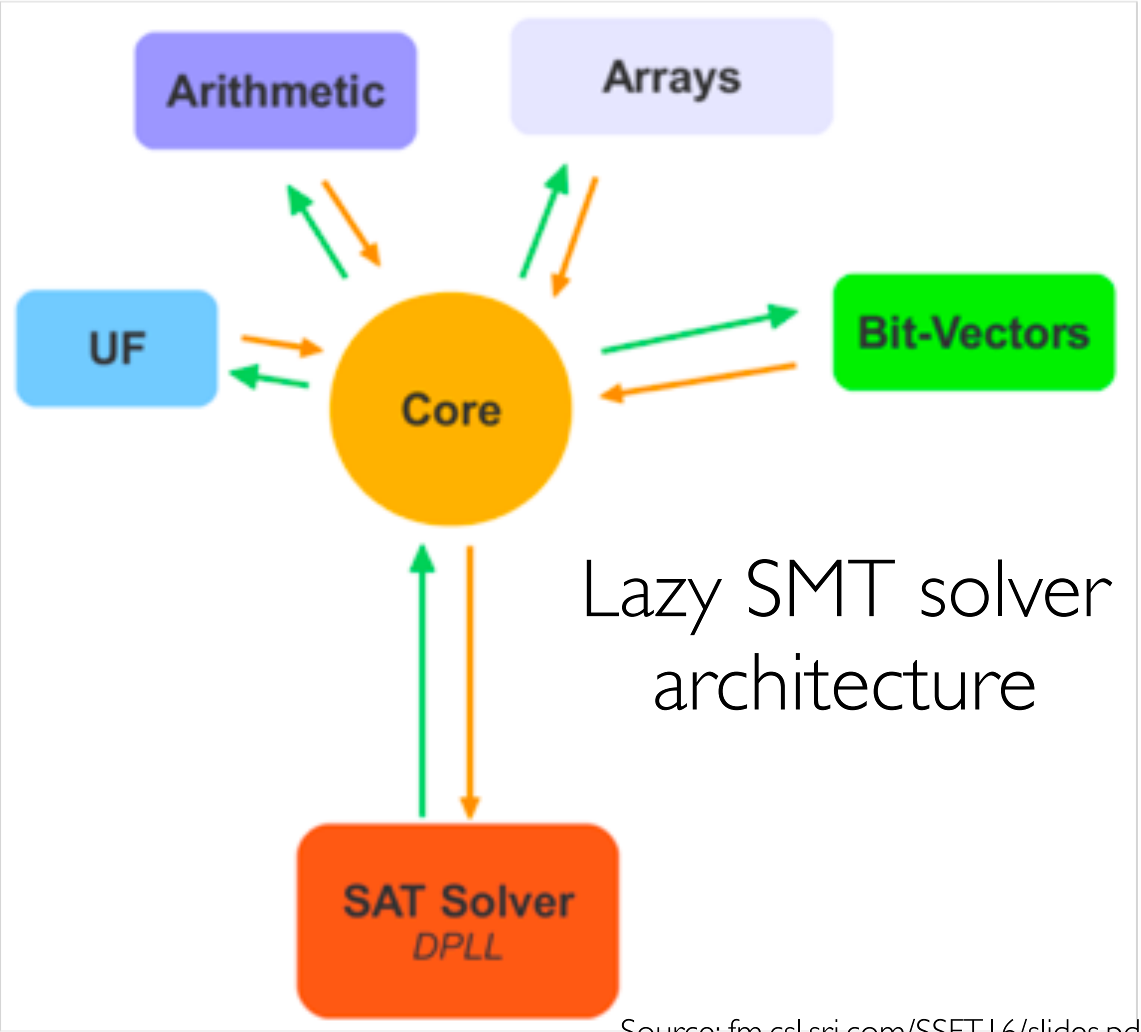
Lazy SMT solver architecture

Source: fm.csl.sri.com/SSFT16/slides.pdf

The virtues of laziness

Eager approach: convert whole SMT formula to SAT formula immediately, then solve with SAT solver

Lazy approach: use *theory solvers*, each specific to a particular theory

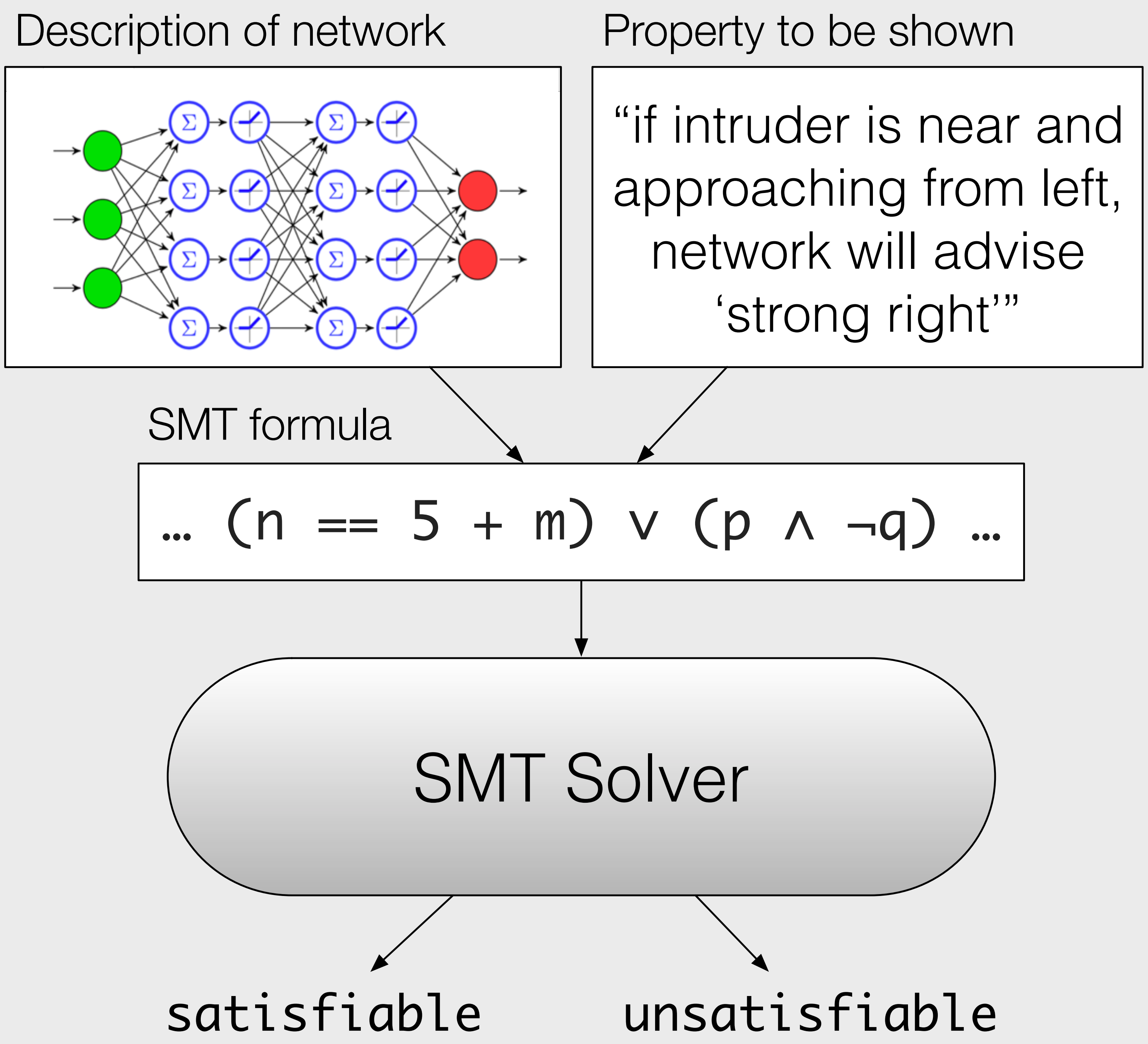


Lazy SMT solver architecture

Source: fm.csl.sri.com/SSFT16/slides.pdf

To exploit domain knowledge and unlock efficiency, we need theory solvers specifically for handling neural networks [SysML '18]

Lazily handling ReLU activations [Katz et al., 2017]



Description of network

Property to be shown

“if intruder is near and approaching from left, network will advise ‘strong right’”

SMT formula

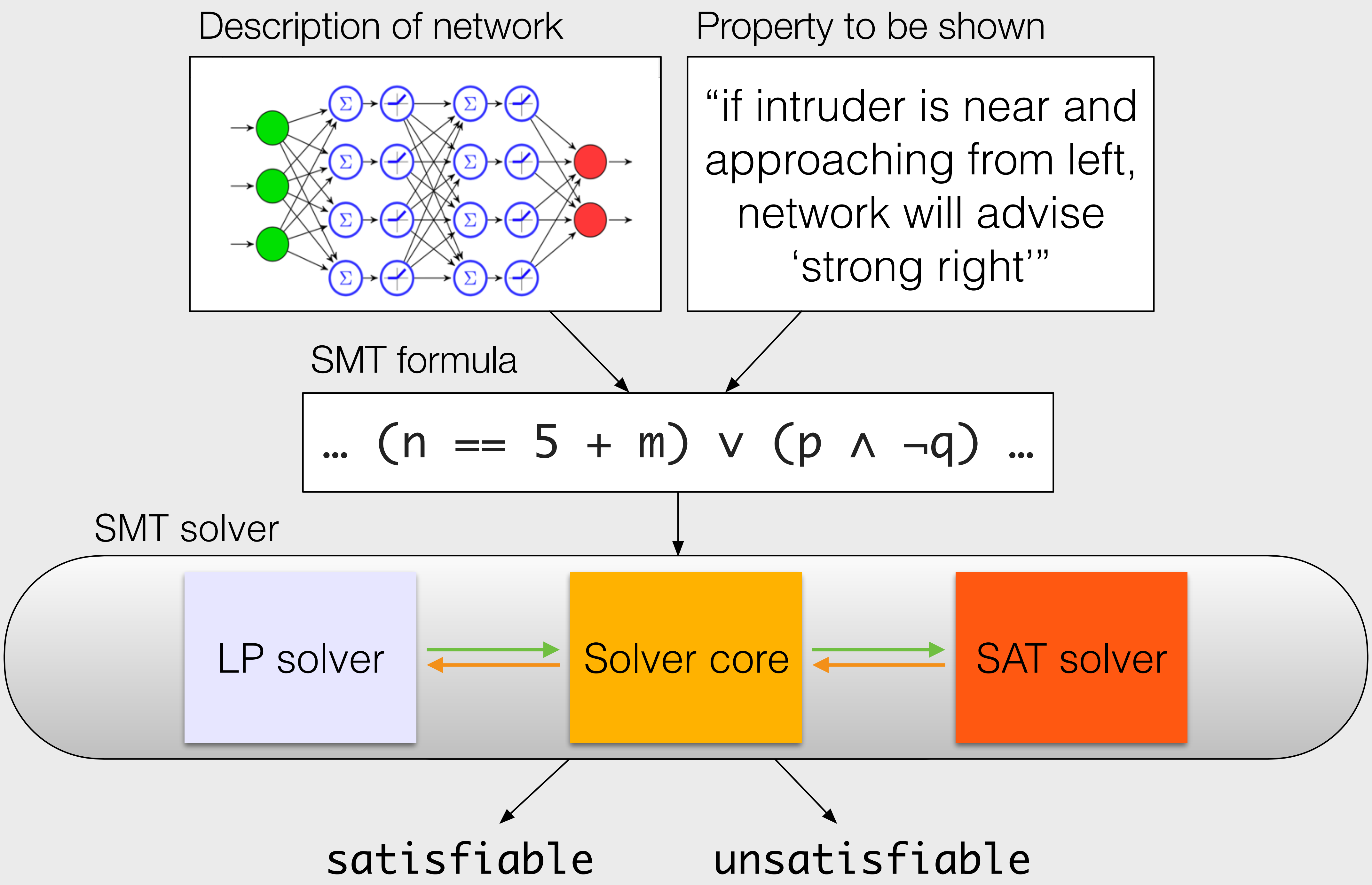
```
… (n == 5 + m) ∨ (p ∧ ¬q) …
```

SMT Solver

```
satisfiable
```

```
unsatisfiable
```

Lazily handling ReLU activations [Katz et al., 2017]



Description of network

Property to be shown

“if intruder is near and approaching from left, network will advise ‘strong right’”

SMT formula

`… (n == 5 + m) ∨ (p ∧ ¬q) …`

SMT solver

LP solver

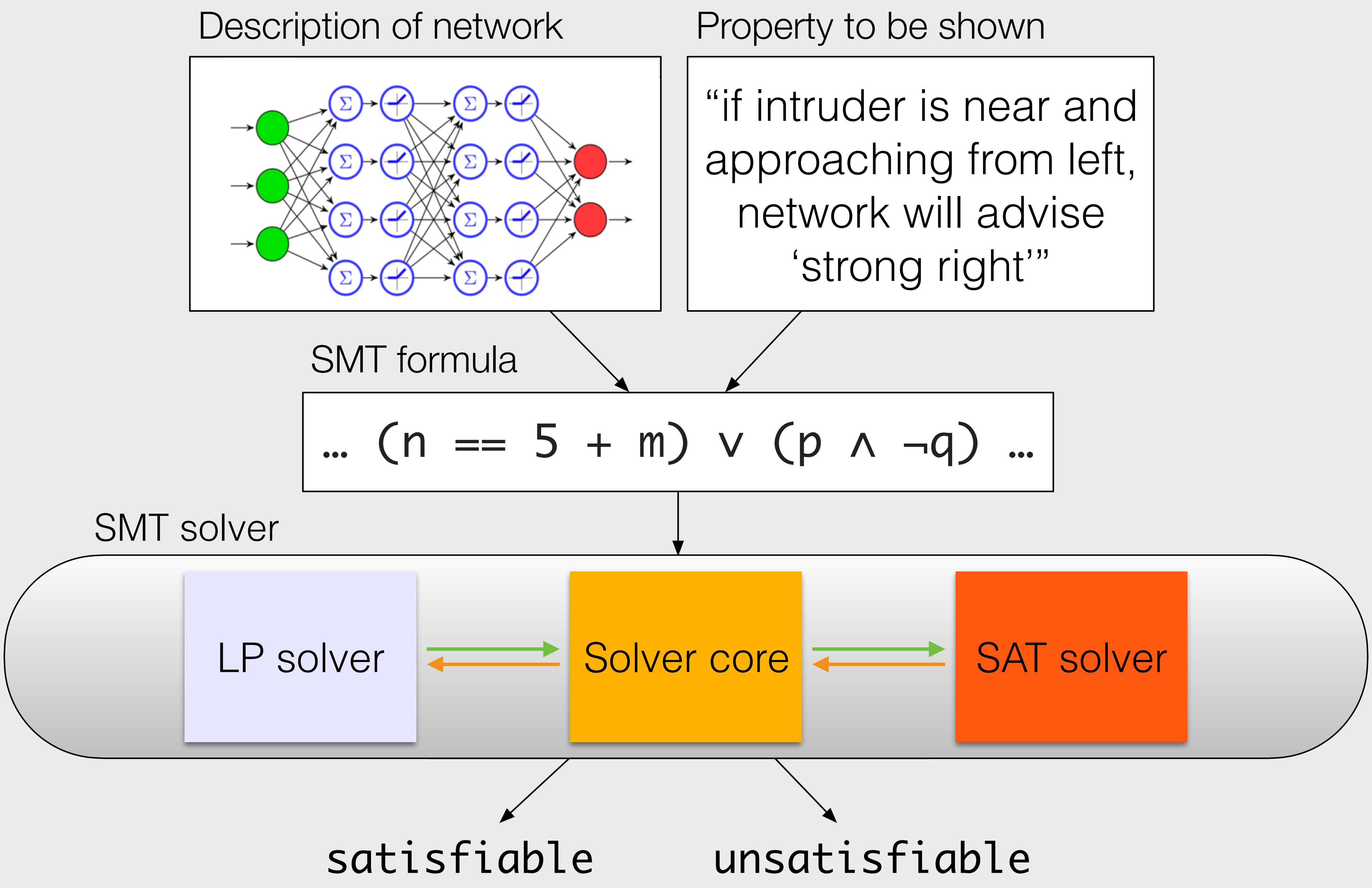
Solver core

SAT solver

`satisfiable`

`unsatisfiable`

Lazily handling ReLU activations [Katz *et al.*, 2017]



Description of network

Property to be shown

“if intruder is near and approaching from left, network will advise ‘strong right’”

SMT formula

`… (n == 5 + m) ∨ (p ∧ ¬q) …`

SMT solver

LP solver

Solver core

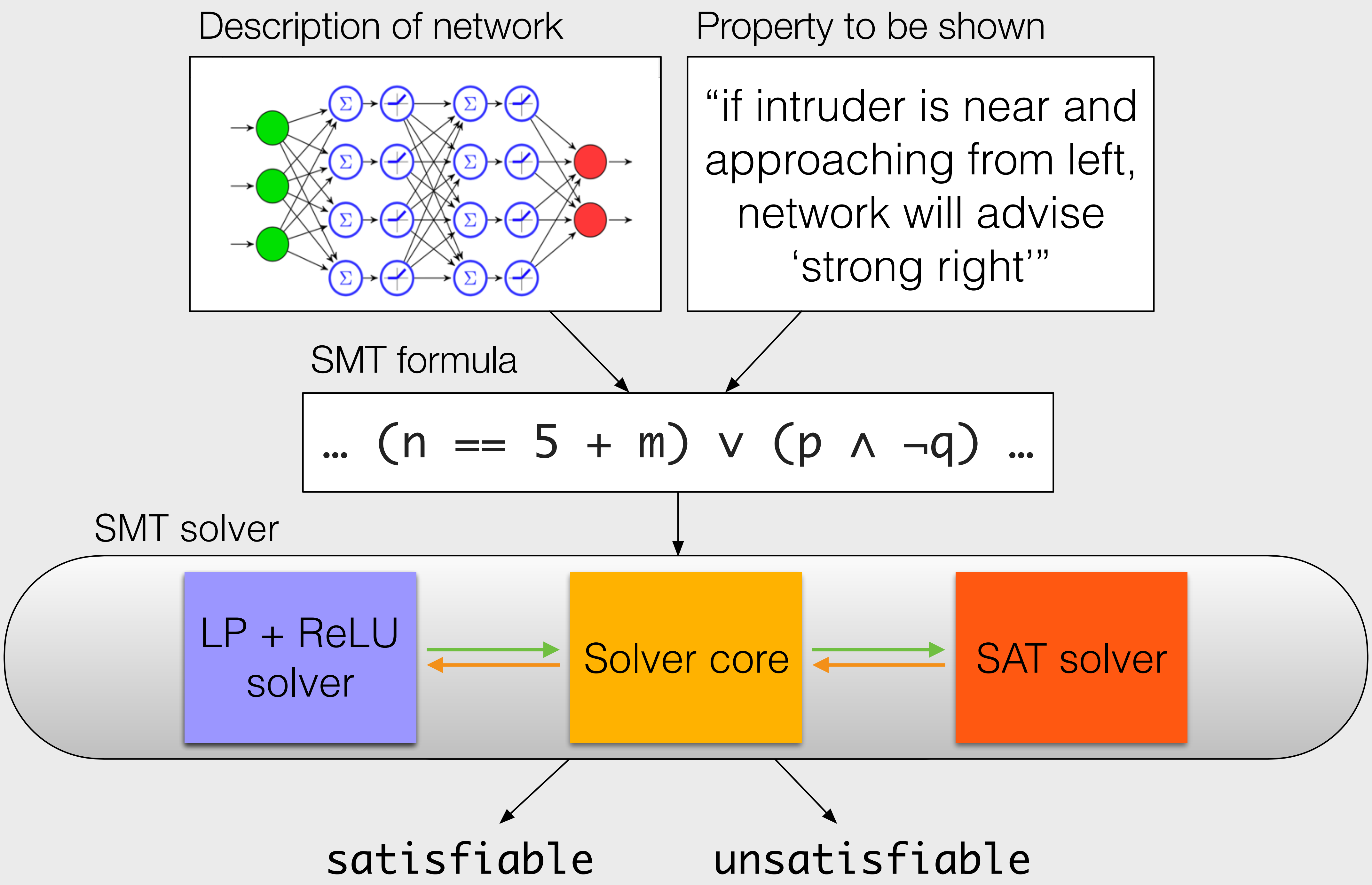
SAT solver

`satisfiable`

`unsatisfiable`

ReLU constraints like `x = max(0, y)` can only be encoded as disjunctions!

Lazily handling ReLU activations [Katz *et al.*, 2017]



Description of network

Property to be shown

"if intruder is near and approaching from left, network will advise 'strong right'"

SMT formula

`… (n == 5 + m) ∨ (p ∧ ¬q) …`

SMT solver

LP + ReLU solver

Solver core

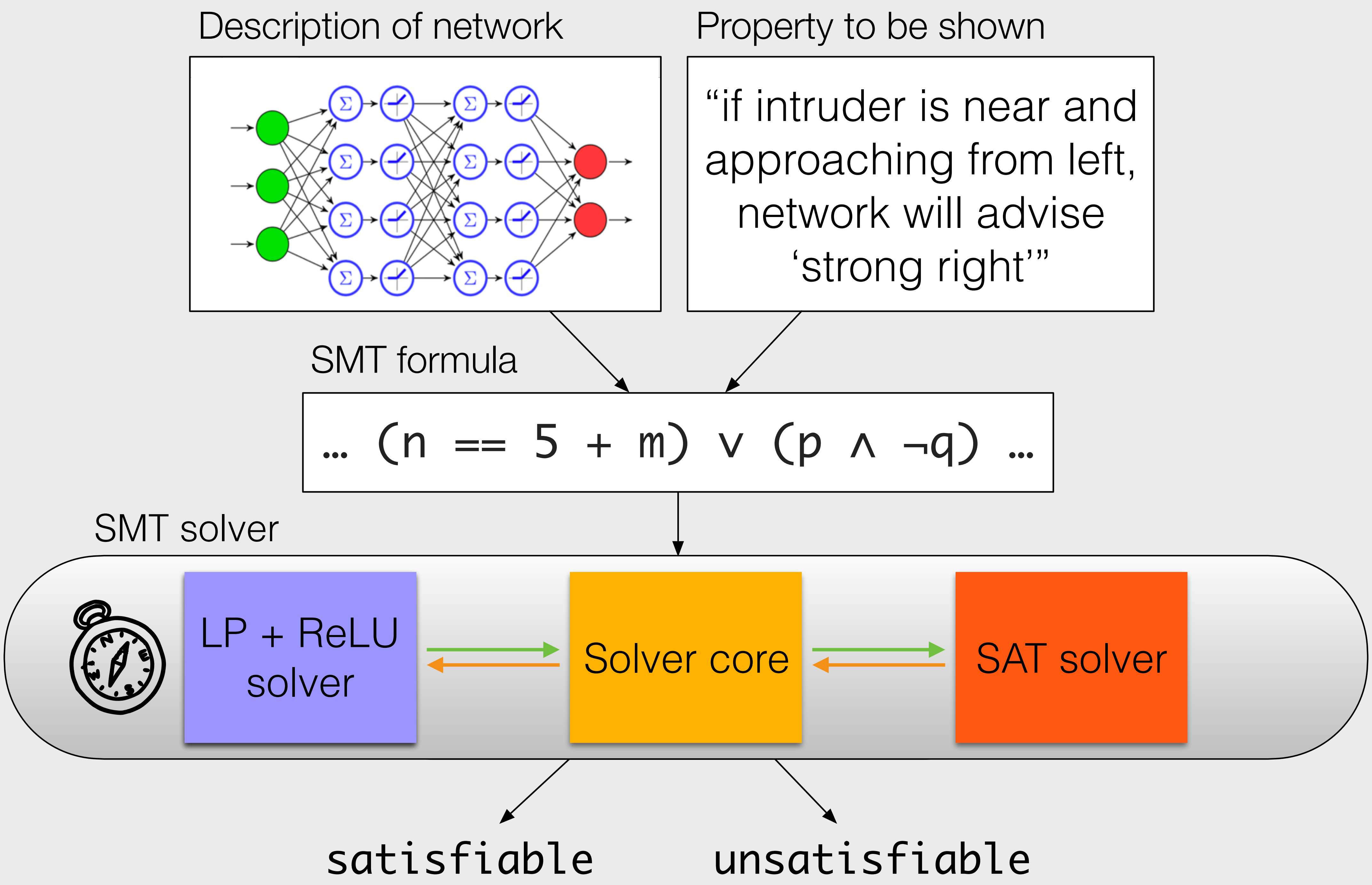
SAT solver

`satisfiable`

`unsatisfiable`

ReLU constraints like **`x = max(0, y)`** can only be encoded as disjunctions!

Lazily handling ReLU activations [Katz *et al.*, 2017]



Description of network

Property to be shown

"if intruder is near and approaching from left, network will advise 'strong right'"

SMT formula

`… (n == 5 + m) ∨ (p ∧ ¬q) …`

SMT solver

LP + ReLU solver

Solver core

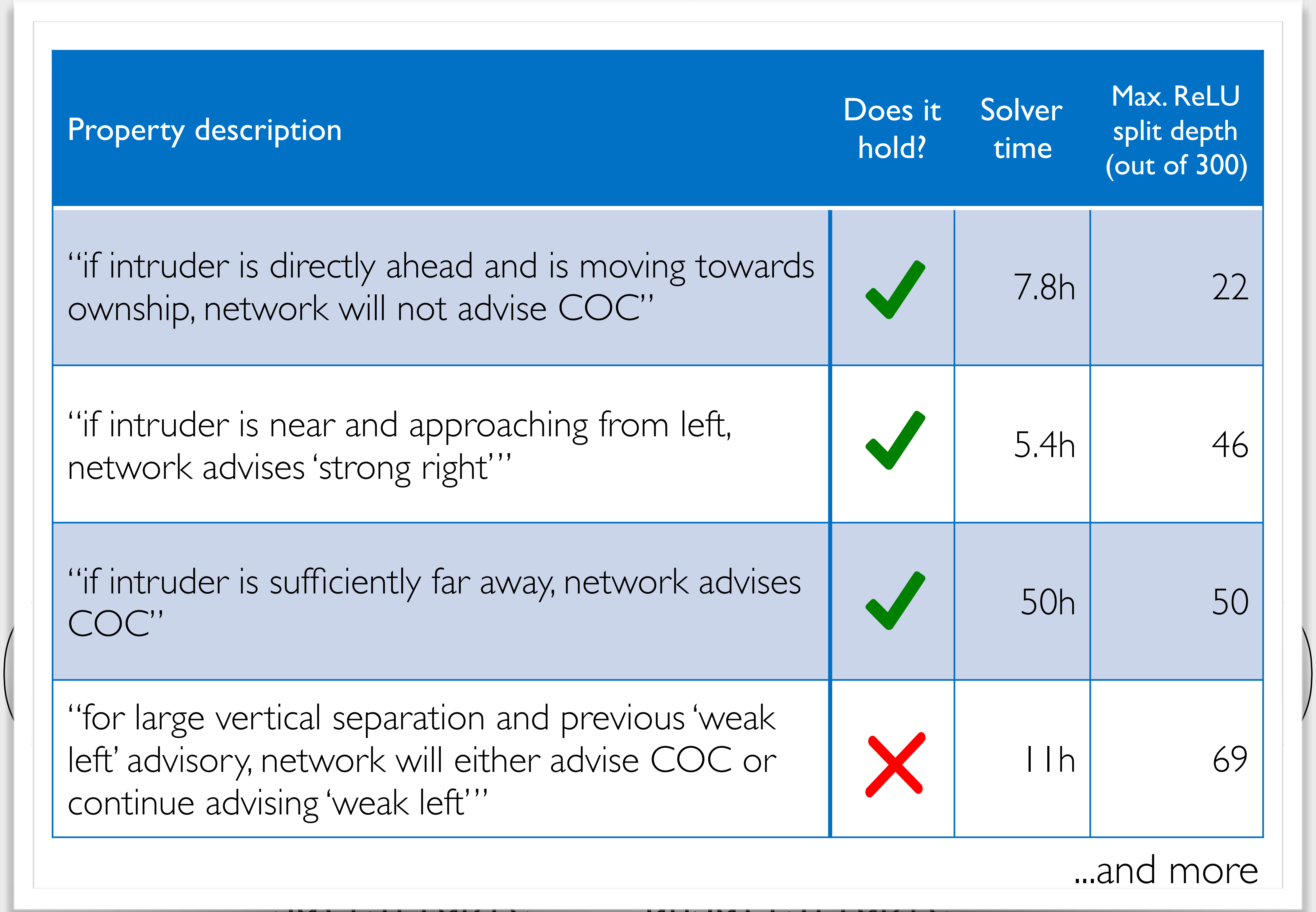
SAT solver

`satisfiable`

`unsatisfiable`

ReLU constraints like **`x = max(0, y)`** can only be encoded as disjunctions!

Lazily handling ReLU activations [Katz *et al.*, 2017]



Property description	Does it hold?	Solver time	Max. ReLU split depth (out of 300)
"if intruder is directly ahead and is moving towards ownship, network will not advise COC"	✓	7.8h	22
"if intruder is near and approaching from left, network advises 'strong right'"	✓	5.4h	46
"if intruder is sufficiently far away, network advises COC"	✓	50h	50
"for large vertical separation and previous 'weak left' advisory, network will either advise COC or continue advising 'weak left'"	✗	11h	69

...and more

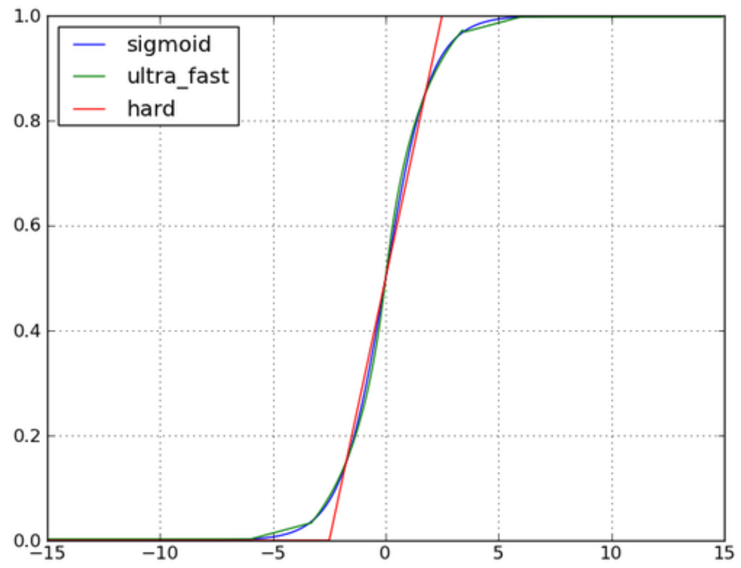
ReLU constraints like `x = max(0, y)` can only be encoded as disjunctions!

Lazily handling ReLU activations [Katz et *al.*, 2017]

Property description	Does it hold?	Solver time	Max. ReLU split depth (out of 300)
“if intruder is directly ahead and is moving towards ownship, network will not advise COC”	✓	7.8h	22
“if intruder is near and approaching from left, network advises ‘strong right’”	✓	5.4h	46
“if intr COC’			
“for la left’ ac contin			

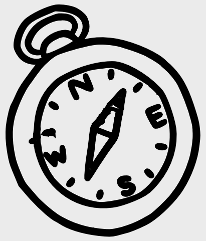
...and more

Extension:
Handle non-ReLU activations using piecewise-linear approximations
[SysML ’18]

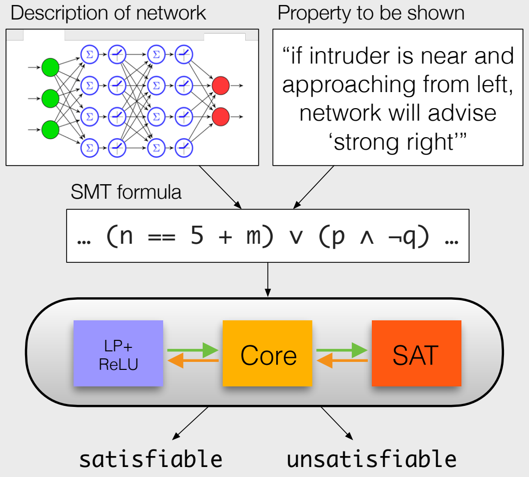


ReLU constraints like `x = max(0, y)` can only be encoded as disjunctions!

A future research agenda



A future research agenda



Description of network

Property to be shown

"if intruder is near and approaching from left, network will advise 'strong right'"

SMT formula

… (n == 5 + m) ∨ (p ∧ ¬q) …

LP+ ReLU

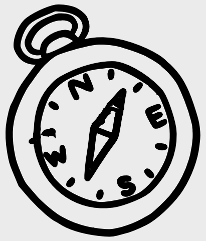
Core

SAT

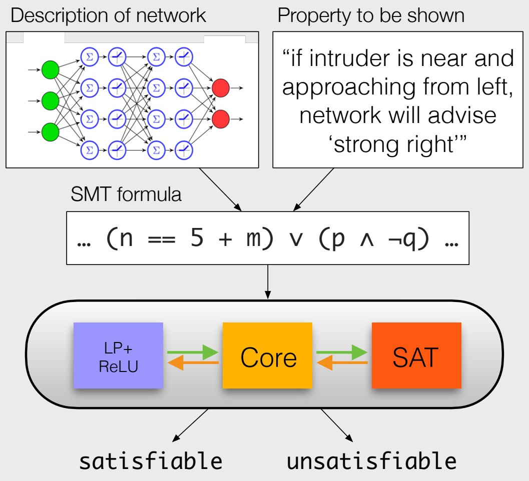
satisfiable

unsatisfiable

Develop new **domain-specific theory solvers** for parallel and distributed computing problems



A future research agenda



Description of network

Property to be shown

“if intruder is near and approaching from left, network will advise ‘strong right’”

SMT formula

`… (n == 5 + m) ∨ (p ∧ ¬q) …`

LP+ ReLU

Core

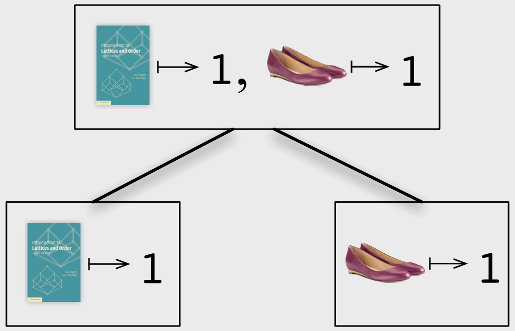
SAT

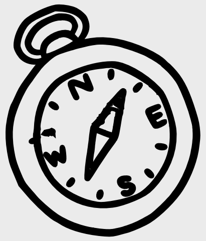
`satisfiable`

`unsatisfiable`

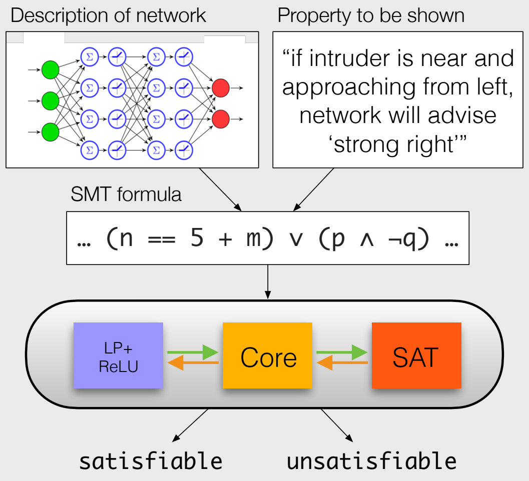
Develop new **domain-specific theory solvers** for parallel and distributed computing problems

Parallelize and distribute SMT solving with strong determinism guarantees





A future research agenda



Description of network

Property to be shown

“if intruder is near and approaching from left, network will advise ‘strong right’”

SMT formula

… (n == 5 + m) ∨ (p ∧ ¬q) …

LP+ ReLU

Core

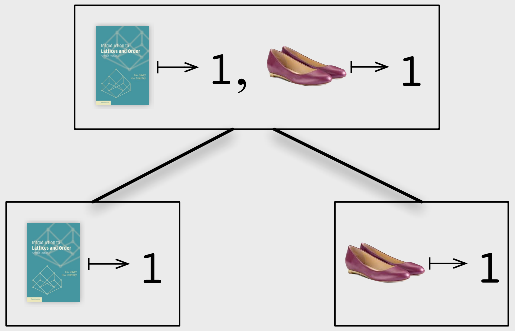
SAT

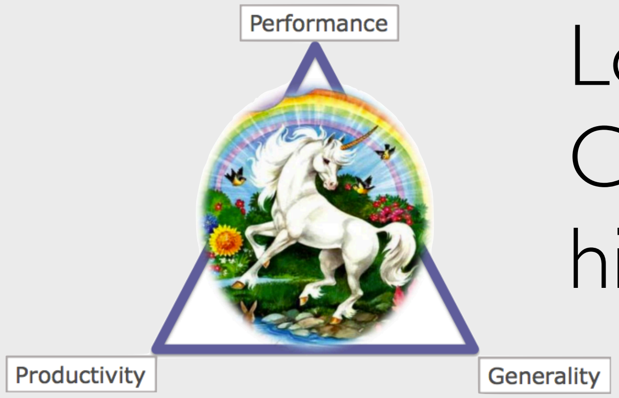
satisfiable

unsatisfiable

Develop new **domain-specific theory solvers** for parallel and distributed computing problems

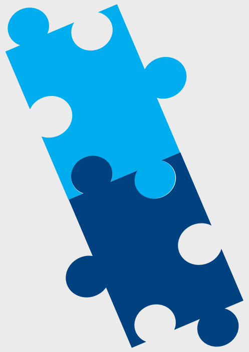
Parallelize and distribute SMT solving with strong determinism guarantees



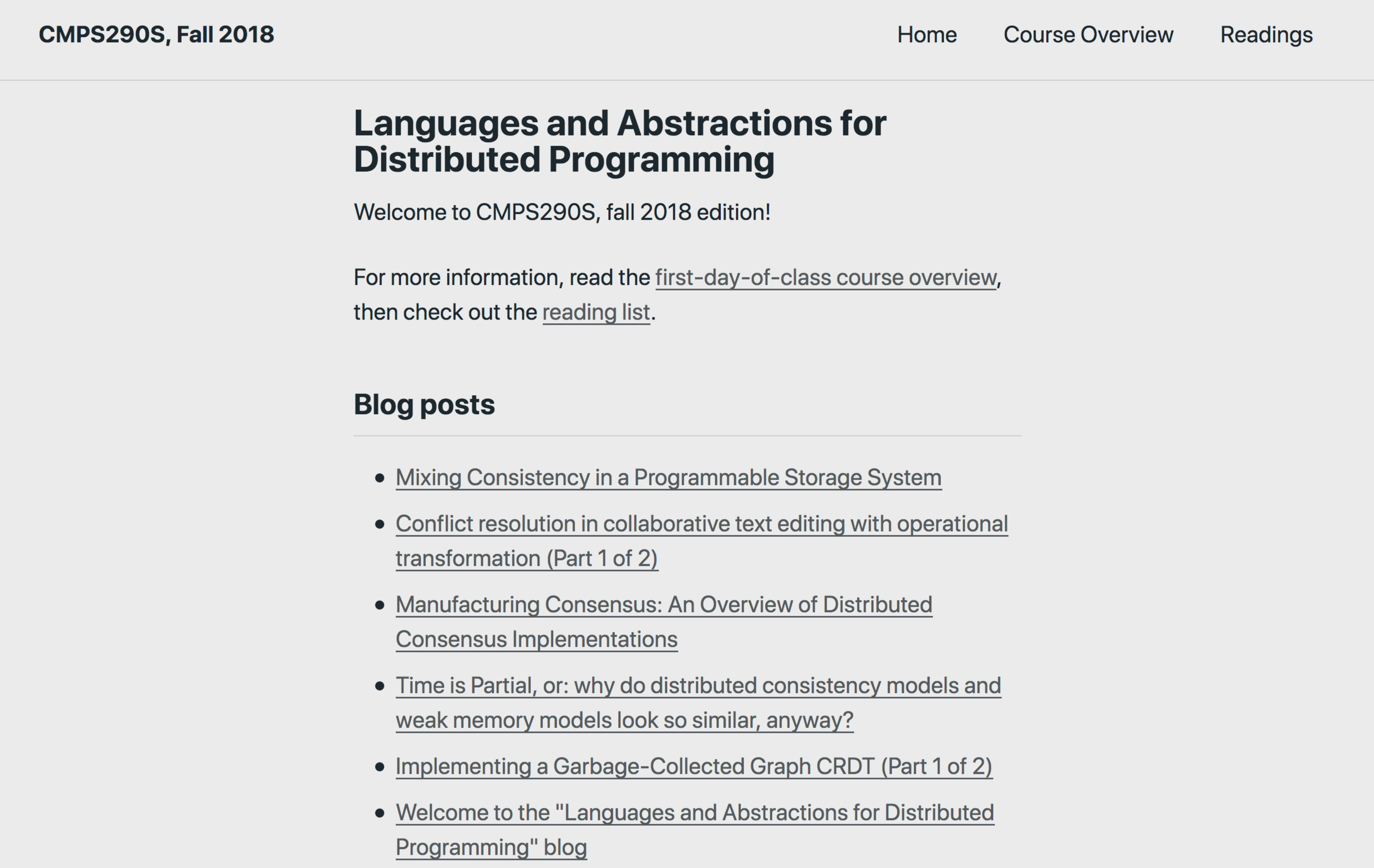


Source: Olukotun *et al*., 2012

Long term: **Democratize solver hacking!** Create tools to make it *really easy* to build high-performance domain-specific solvers



composition.al/CMPS290S-2018-09/



CMPS290S, Fall 2018

Home Course Overview Readings

Languages and Abstractions for Distributed Programming

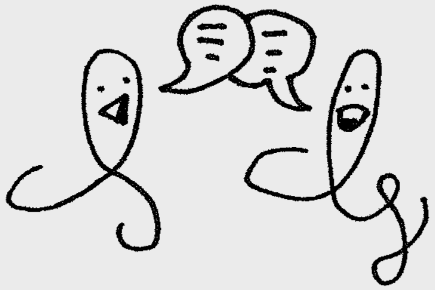
Welcome to CMPS290S, fall 2018 edition!

For more information, read the first-day-of-class course overview, then check out the reading list.

Blog posts

- Mixing Consistency in a Programmable Storage System
- Conflict resolution in collaborative text editing with operational transformation (Part 1 of 2)
- Manufacturing Consensus: An Overview of Distributed Consensus Implementations
- Time is Partial, or: why do distributed consistency models and weak memory models look so similar, anyway?
- Implementing a Garbage-Collected Graph CRDT (Part 1 of 2)
- Welcome to the "Languages and Abstractions for Distributed Programming" blog

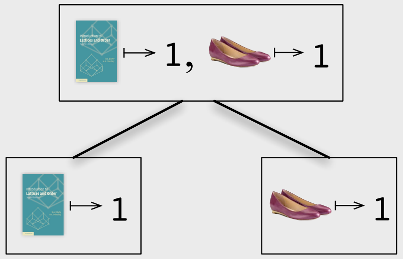
Thank you!



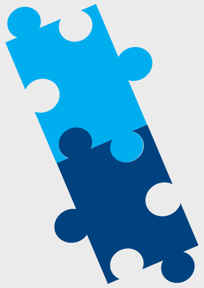
Email: lkuper@ucsc.edu

Papers, etc.: users.soe.ucsc.edu/~lkuper/

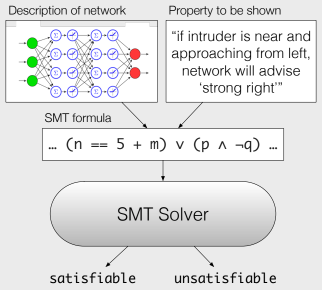
Research blog: composition.al



Lattice-based data structures (LVars) for deterministic programming
[POPL '14, PLDI '14, FHPC '13, WoDet '14]



Non-invasive DSLs for productive parallelism
[ECOOP '17]



Description of network

Property to be shown

"if intruder is near and approaching from left, network will advise 'strong right'"

SMT formula

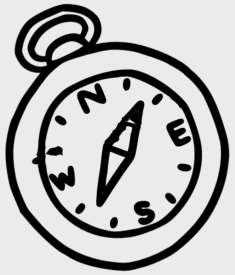
… (n == 5 + m) ∨ (p ∧ ¬q) …

SMT Solver

satisfiable

unsatisfiable

SMT-based verification of safety-critical neural networks
[SysML '18]



Guiding principle:
Find the right **high-level abstractions** to enable **efficient** computation

Special thanks to Jason Reed for drawings!