Beyond Reduction ...

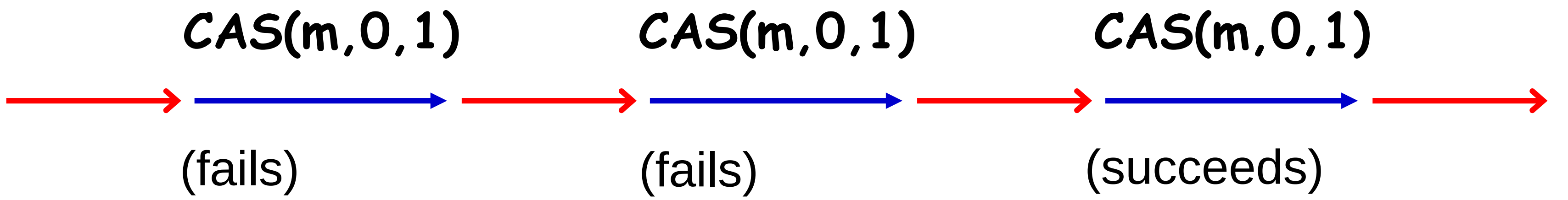
Busy Acquire

```
atomic void busy_acquire() {
  while (true) {
    if (CAS(m,0,1)) break;
  }
}
```

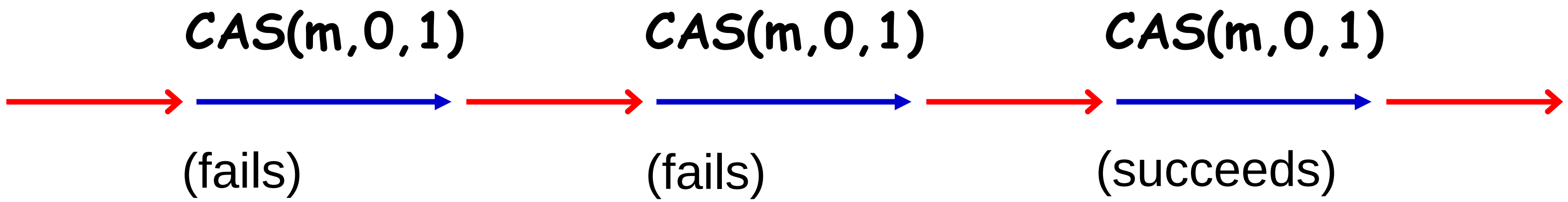
```
if (m == 0) {
  m = 1; return true;
} else {
  return false;
}
```

Busy Acquire

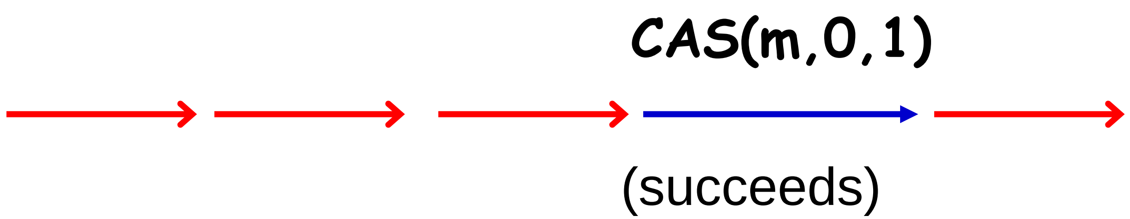
```
atomic void busy_acquire() {
  while (true) {
    if (CAS(m,0,1)) break;
  }
}
```



- Non-Serial Execution:



- Serial Execution:



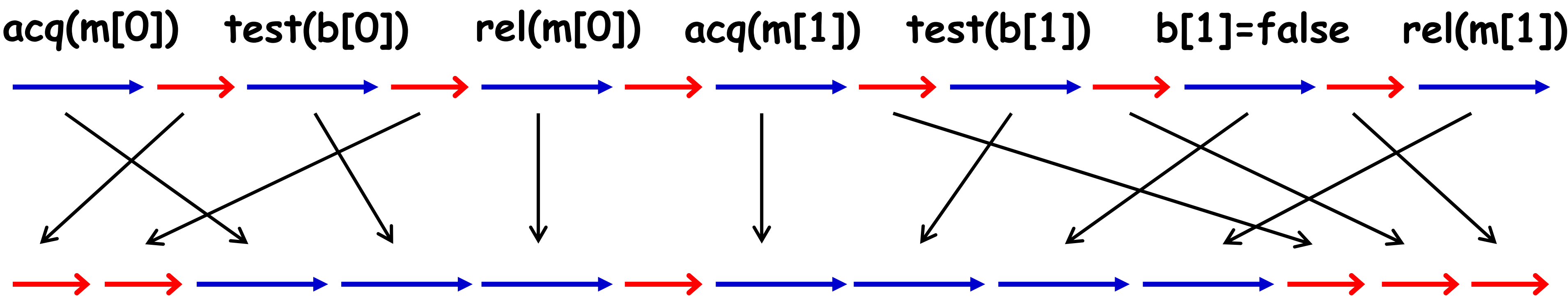
- Atomic but not reducible

alloc

```
boolean b[MAX]; // b[i]==true iff block i is free
Lock m[MAX];

atomic int alloc() {
  int i = 0;
  while (i < MAX) {
    acquire(m[i]);
    if (b[i]) {
      b[i] = false;
      release(m[i]);
      return i;
    }
    release(m[i]);
    i++;
  }
  return -1;
}
```

alloc



alloc is not Atomic

- There are non-serial executions with no equivalent serial executions

```
m[0] = m[1] = 0; b[0] = b[1] = false;

t = alloc();   ||  free(0); free(1);
```

```
void free(int i) {
  acquire(m[i]);
  b[i] = true;
  release(m[i]);
}
```

```
m[0] = m[1] = 0; b[0] = b[1] = false;

t = alloc();   ||  free(0); free(1);
```

- Non-Serial Execution:

loop for b[0] → free(0) → free(1) → loop for b[1] → t = 1

- Serial Executions:

loop for b[0] → loop for b[1] → free(0) → free(1) → t = -1

free(0) → free(1) → loop for b[0] → t = 0

free(0) → loop for b[0] → free(1) → t = 0

Extending Atomicity

- Atomicity doesn't always hold for methods that are "intuitively atomic"
  - serializable but not reducible (busy_acquire)
  - not serializable (alloc)
- Examples
  - initialization
  - resource allocation
  - wait/notify
  - caches
  - commit/retry transactions

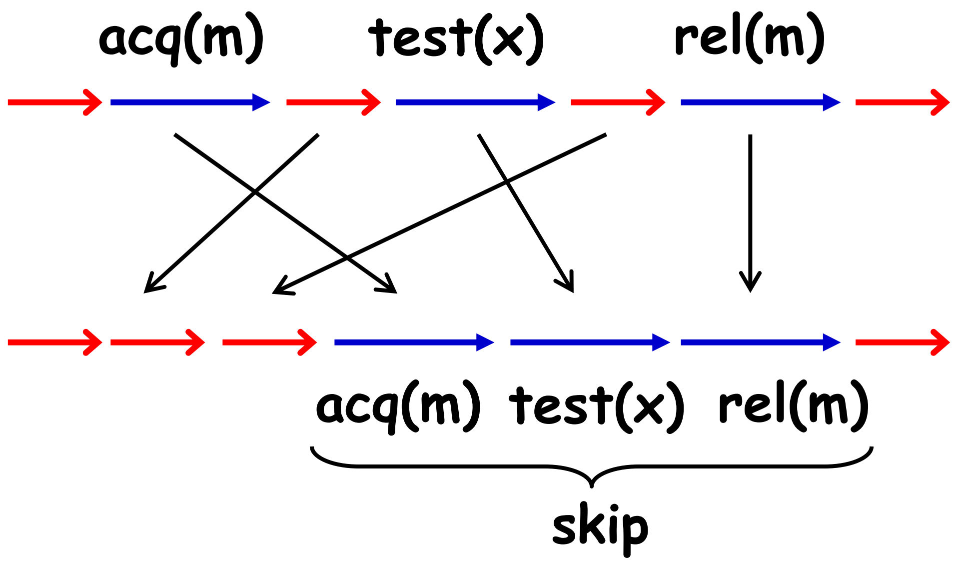
Pure Code Blocks

- Pure block: **`pure { E }`**
  - If **E** terminates normally, it does not update state visible outside of **E**
  - **E** is reducible
- Example

```
while (true) {
  pure {
    acquire(mx);
    if (x == 0) { x = 1; release(mx); break; }
    release(mx);
  }
}
```

Purity and Abstraction

- *A* **pure** block's behavior under normal termination is the same as **skip**



- Abstract execution semantics:
  - **pure** blocks can be skipped

Abstraction

- *Abstract semantics* that admits more behaviors
  - pure blocks can be skipped
  - hides "irrelevant" details (ie, failed loop iters)
  -
- Program must still be (sequentially) correct in abstract semantics

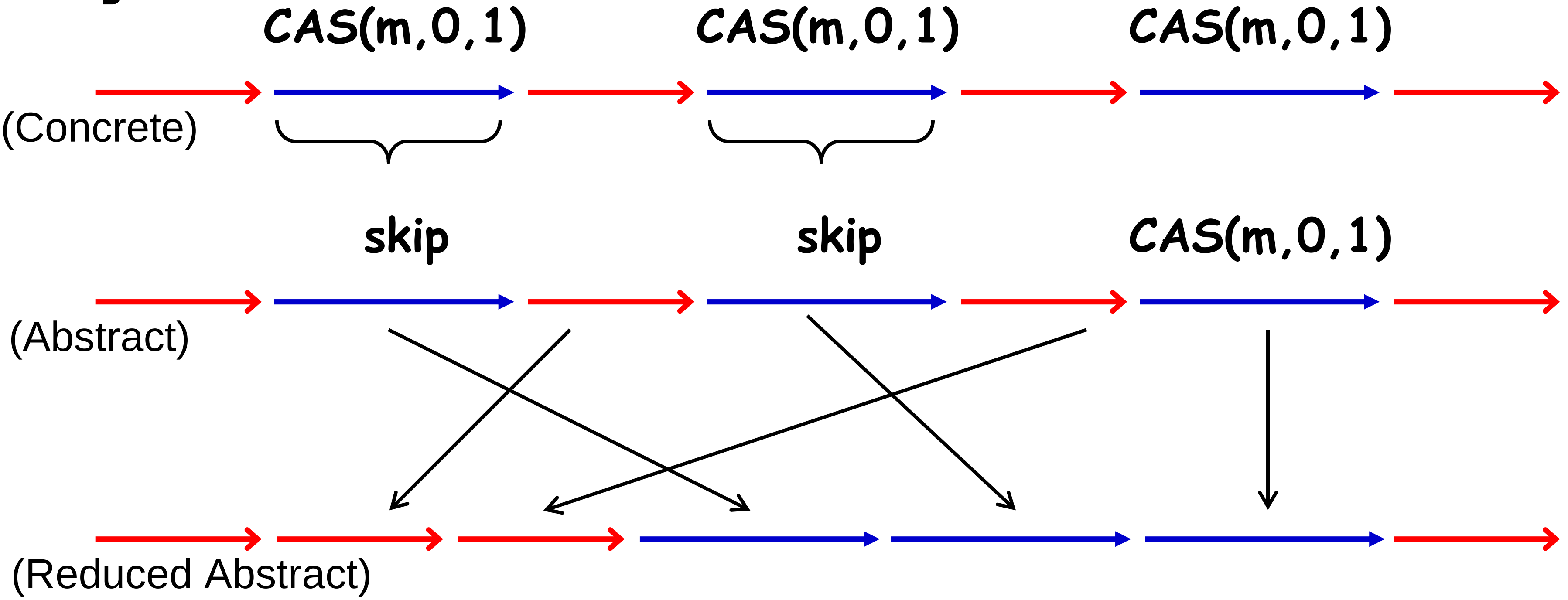
- Abstract semantics make reduction possible

Busy Acquire

```
atomic void busy_acquire() {
  while (true) {
    pure { if (CAS(m,0,1)) break; }
  }
}
```

Abstract Execution of Busy Acquire

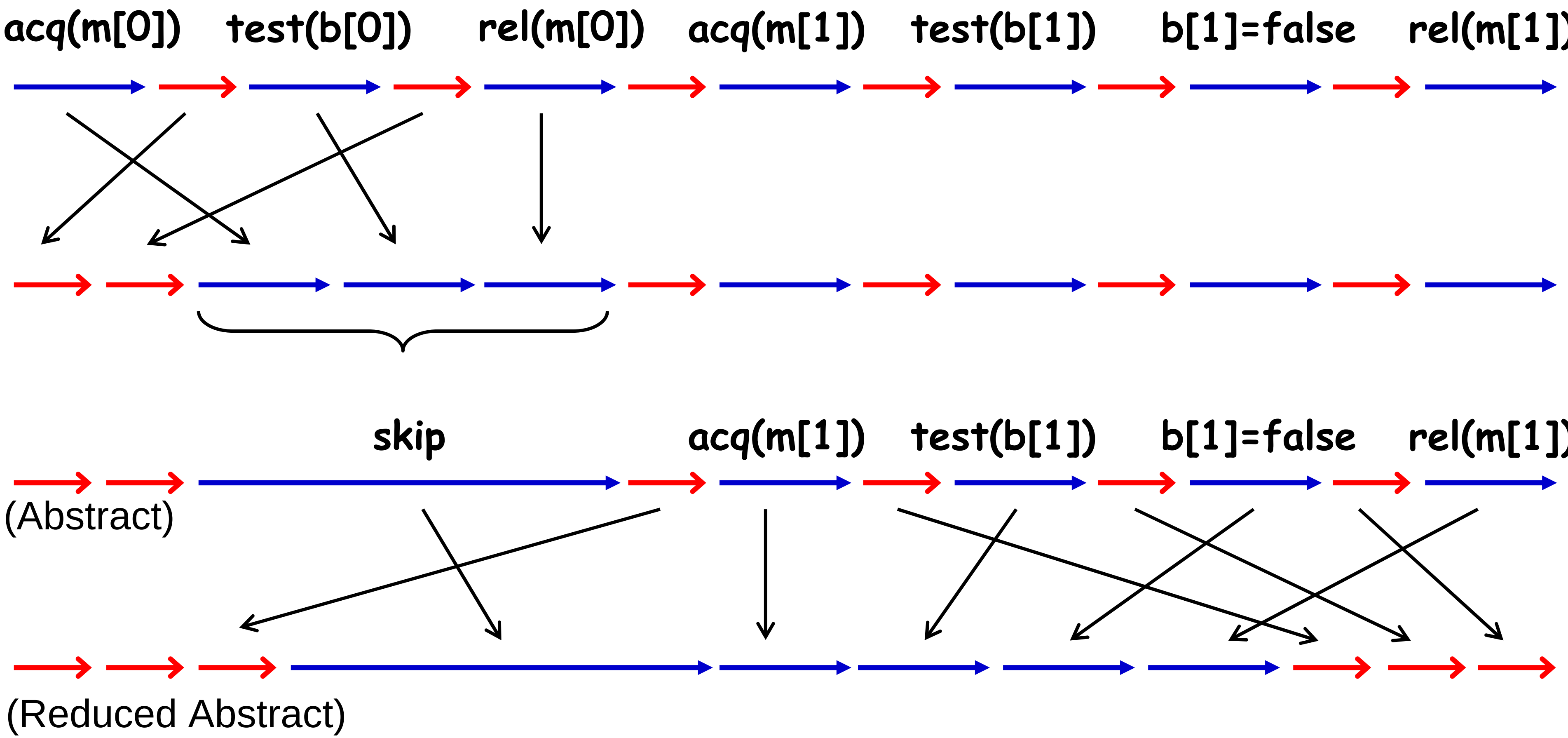
```
atomic void busy_acquire() {
  while (true) {
    pure { if (CAS(m,0,1)) break; }
  }
}
```



alloc

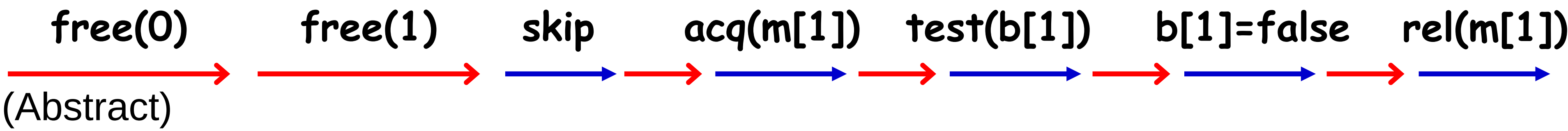
```
atomic int alloc() {
  int i = 0;
  while (i < MAX) {
    pure {
      acquire(m[i]);
      if (b[i]) {
        b[i] = false;
        release(m[i]);
        return i;
      }
      release(m[i]);
    }
    i++;
  }
  return -1;
}
```

Abstract Execution of alloc



Abstraction

- Abstract semantics admits more executions



- Can still reason about important properties
  - "**alloc** returns either the index of a freshly allocated block or -1"
  - cannot guarantee "**alloc** returns smallest possible index"
    - but what does this really mean anyway???

To Atomicity and Beyond ...

Commit-Atomicity

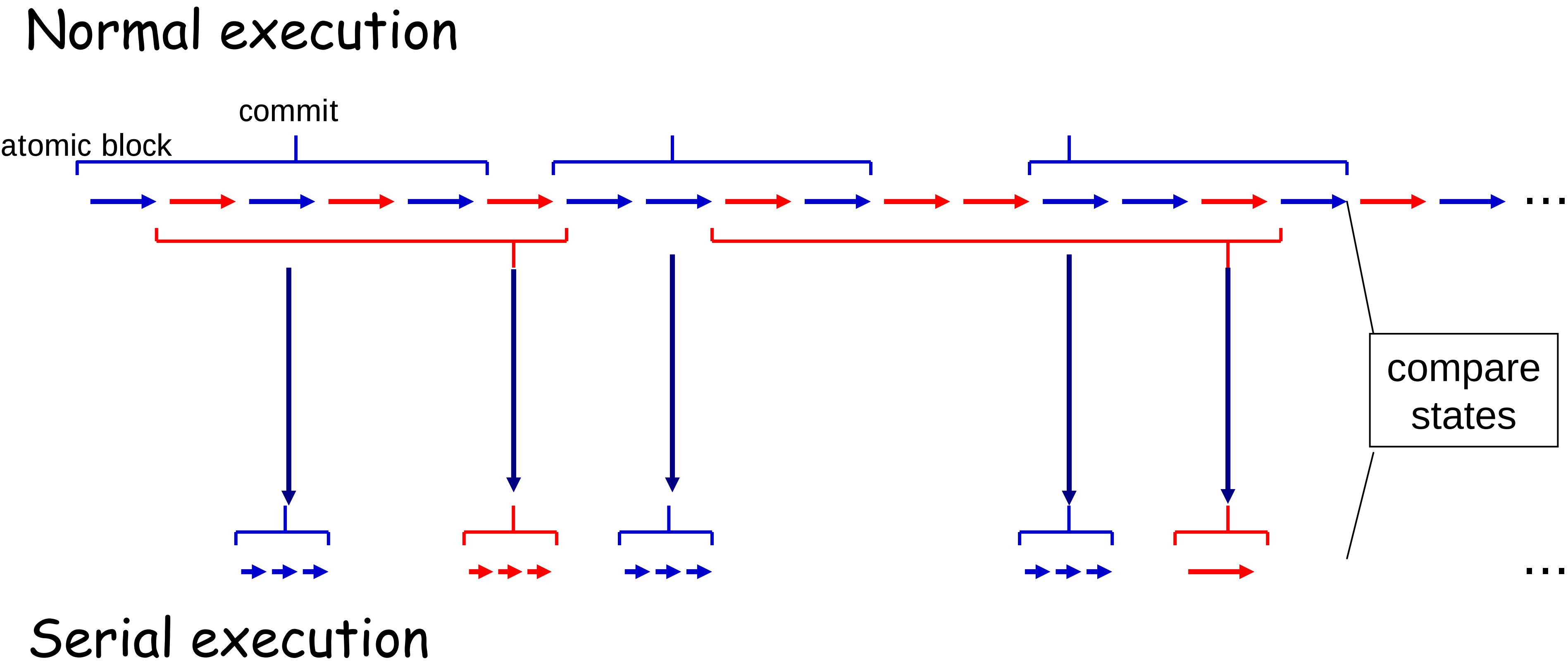
- Reduction
  - Great if can get serial execution via commuting
- Reduction + Purity
  - Great if non-serial execution performs extra *pure* loops

- Commit Atomicity
  - More heavyweight technique to verify if some corresponding serial execution has same behavior
    - can take different steps

Checking Commit Atomicity

- Run *normal* and *serial* executions of program concurrently, on separate stores
- Normal execution runs as normal
  - threads execute atomic blocks
  - each atomic block has *commit* point
- Serial execution
  - runs on separate *shadow* store
  - when normal execution *commits* an atomic block, serial execution runs entire atomic block serially
- Check two executions yield same behavior

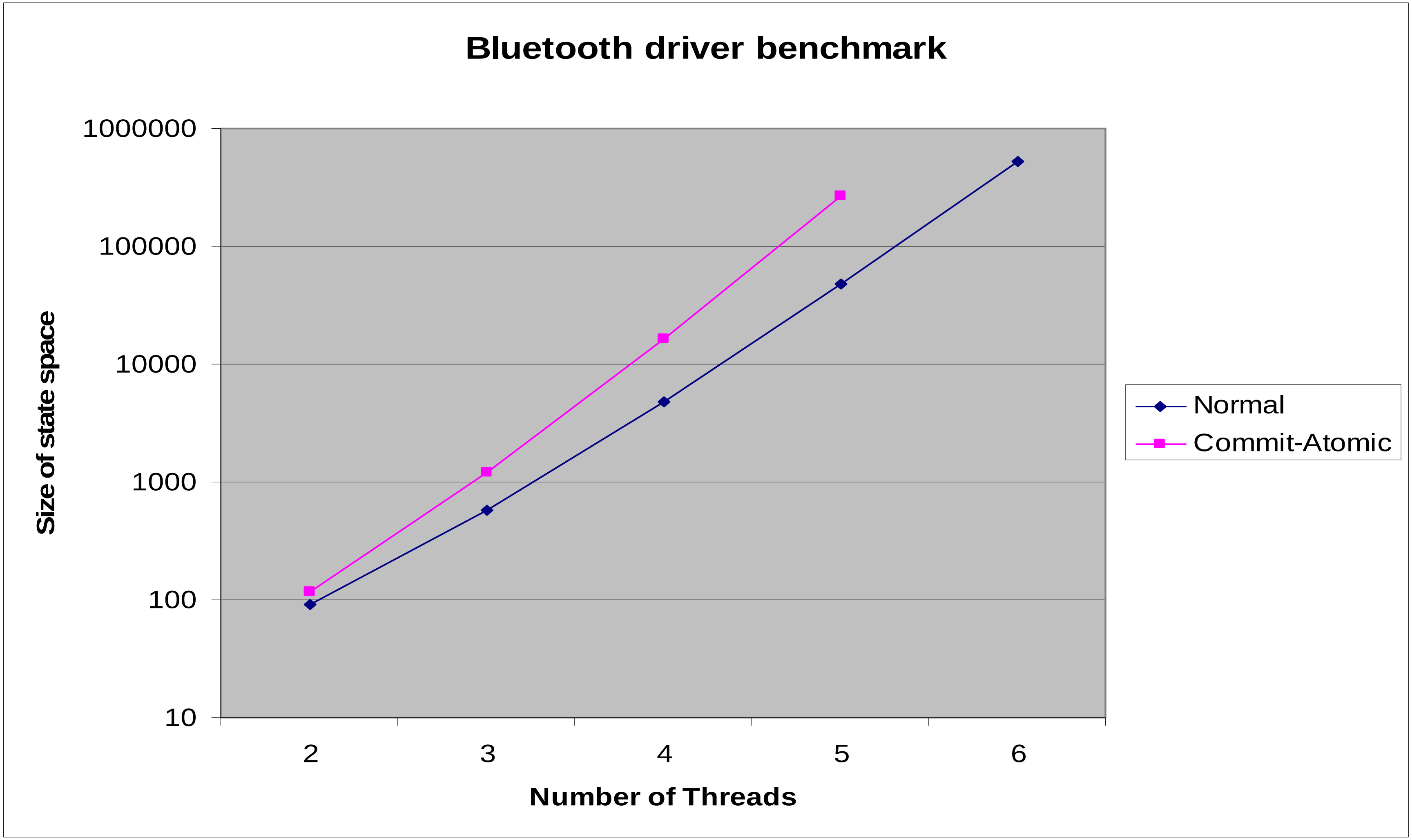
Commit-Atomic



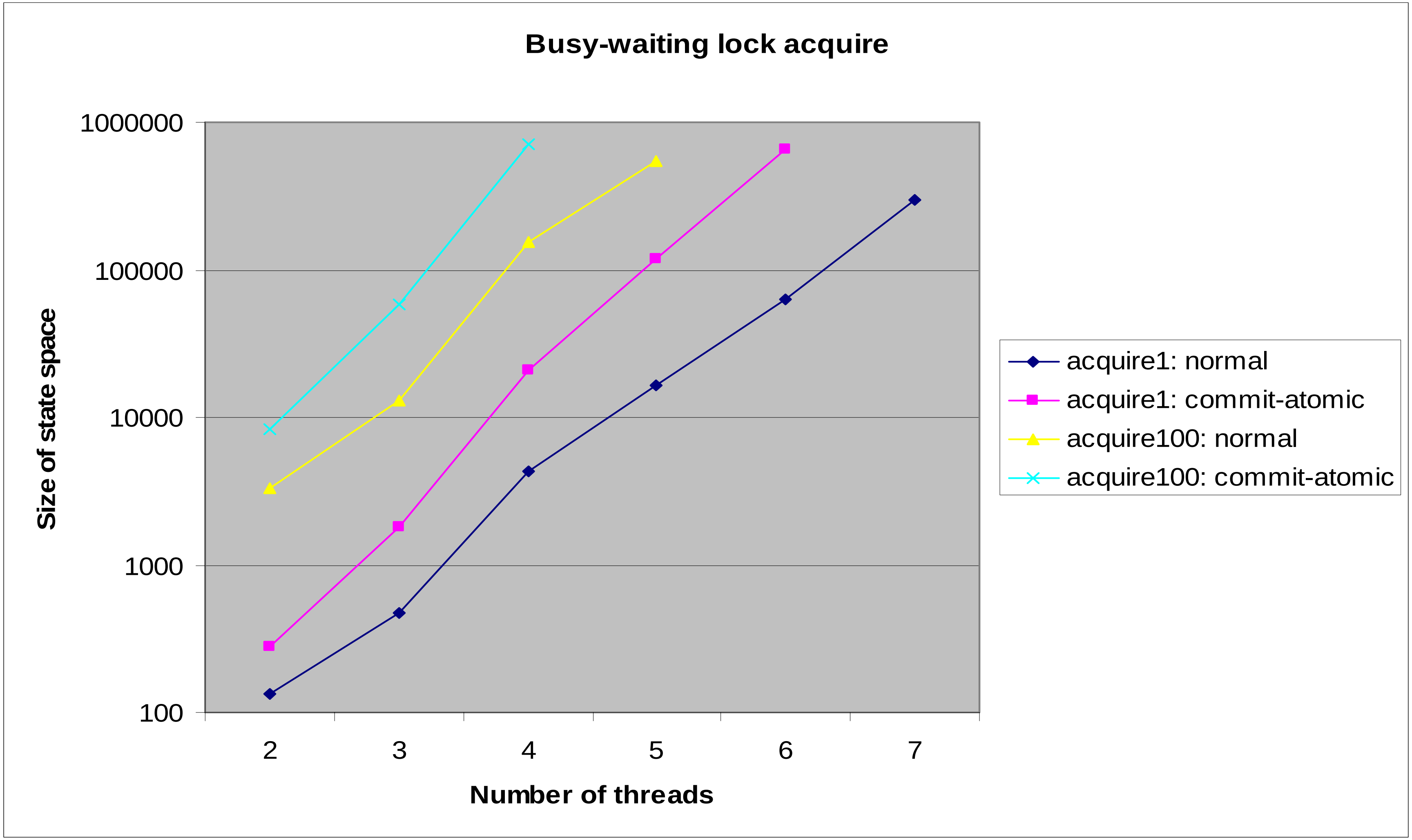
Preliminary Evaluation

- Some small benchmarks
  - Bluetooth device driver
    - atomicity violation due to error
  - Busy-waiting lock acquire
    - acquire1: 1 line of code in critical section
    - acquire100: 100 lines of code in critical section
- Hand translated to PROMELA code
  - Two versions, with and without commit-atomic
  - Model check with SPIN

Performance: Bluetooth device driver



Performance: acquire1 and acquire100



Summary

- Atomicity
  - concise, semantically deep partial specification
  - aka serializability
- Reduction
  - lightweight technique for verifying atomicity
  - can verify with types, or dynamically
  - plus purity, for complex cases
- Commit-Atomicity
  - more general technique

Summary

- Atomicity
  - concise, semantically deep partial specification
- Reduction
  - lightweight technique for verifying atomicity
- Commit-Atomicity
  - more general technique
- Future work
  - combine reduction and commit-atomic
  - generalizing atomicity
    - *temporal logics for determinism?*