Composable Specifications for Structured Shared-Memory Communication

Benjamin P. Wood, Adrian Sampson, Luis Ceze, Dan Grossman

University of Washington

Code-Communication Specifications

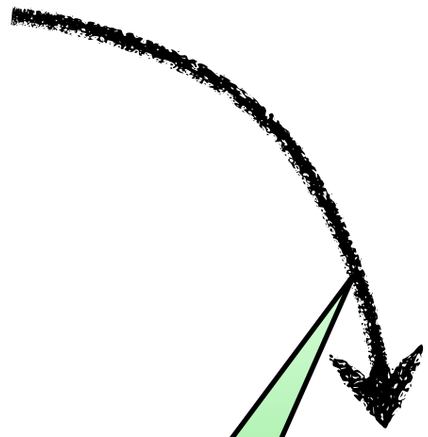
Writer Thread	Reader Thread
`enqueue(...);`	
	`dequeue();`

Code-Communication Specifications

Writer Thread

Reader Thread

```
enqueue(...);
```



```
dequeue();
```

May writes in **`enqueue`** be read by other threads in **`dequeue`**?

Code-Communication Specifications

Writer Thread

Reader Thread

What **code** may communicate across threads?

May writes in `enqueue` be read by other threads in `dequeue`?

Code-Communication Specifications

Writer Thread

Reader Thread

What **code** may communicate across threads?

`enqueue` ✔ `dequeue`

May writes in `enqueue` be read by other threads in `dequeue`?

Code-Communication Specifications

Writer Thread

Reader Thread

What **code** may communicate across threads?

`enqueue` ✔ `dequeue`

`enqueue` ✘ `render`

May writes in `enqueue` be read by other threads in `dequeue`?

Implicitly Shared Memory

```
this.buffer[...] = i;

this.size = this.size + 1;
```

Implicitly Shared Memory

```
this.buffer[...] = i;
```

What is shared? What is not?

```
this.size = this.size + 1;
```

Implicitly Shared Memory

```
this.buffer[...] = i;
```

What is shared? What is not?

```
this.size = this.size + 1;
```

Thread-private?

Implicitly Shared Memory

Read-only?

```
this.buffer[...] = i;
```

What is shared? What is not?

```
this.size = this.size + 1;
```

Thread-private?

Implicitly Shared Memory

Read-only?

```
this.buffer[...] = i;
```

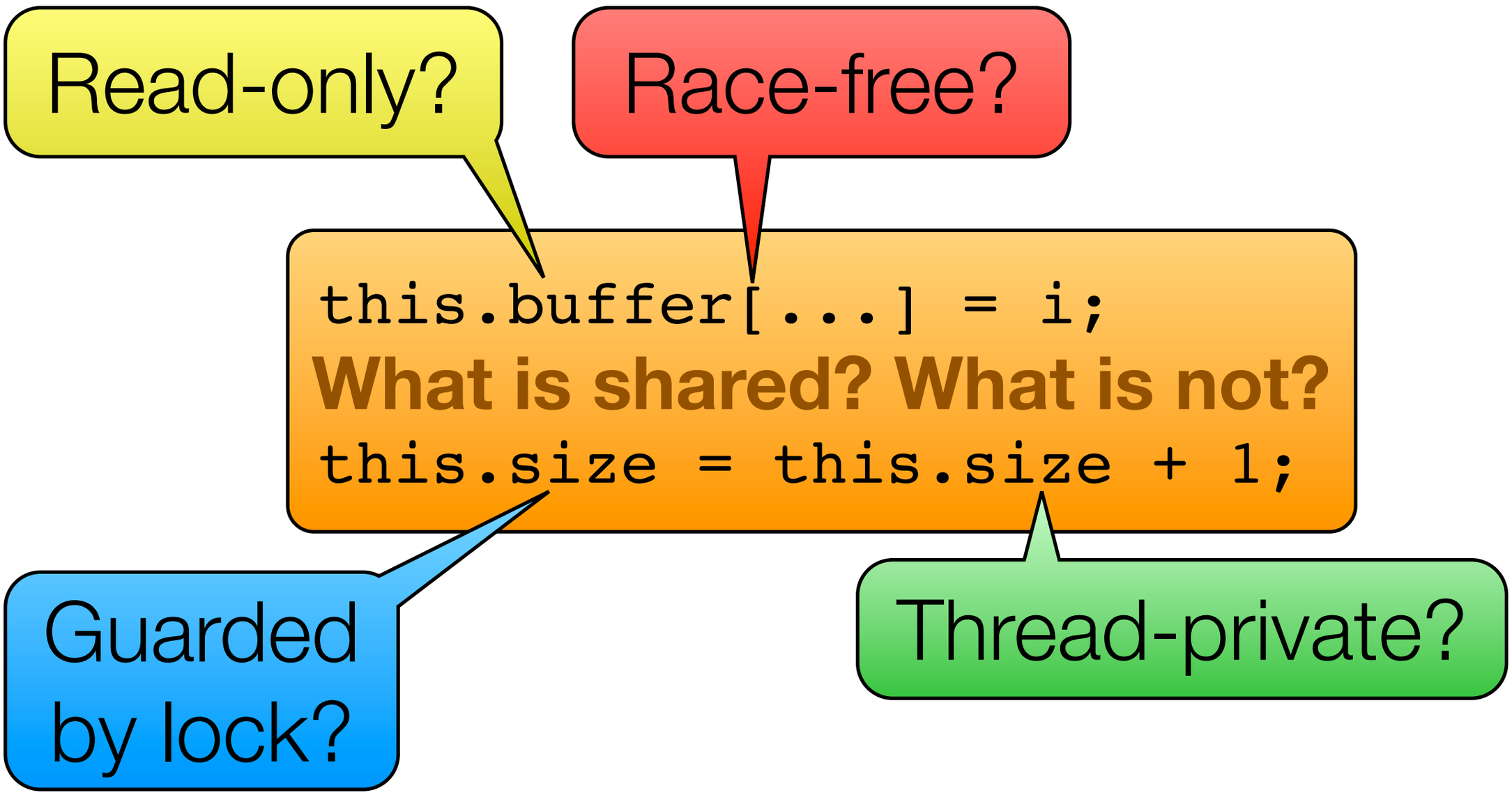
What is shared? What is not?

```
this.size = this.size + 1;
```

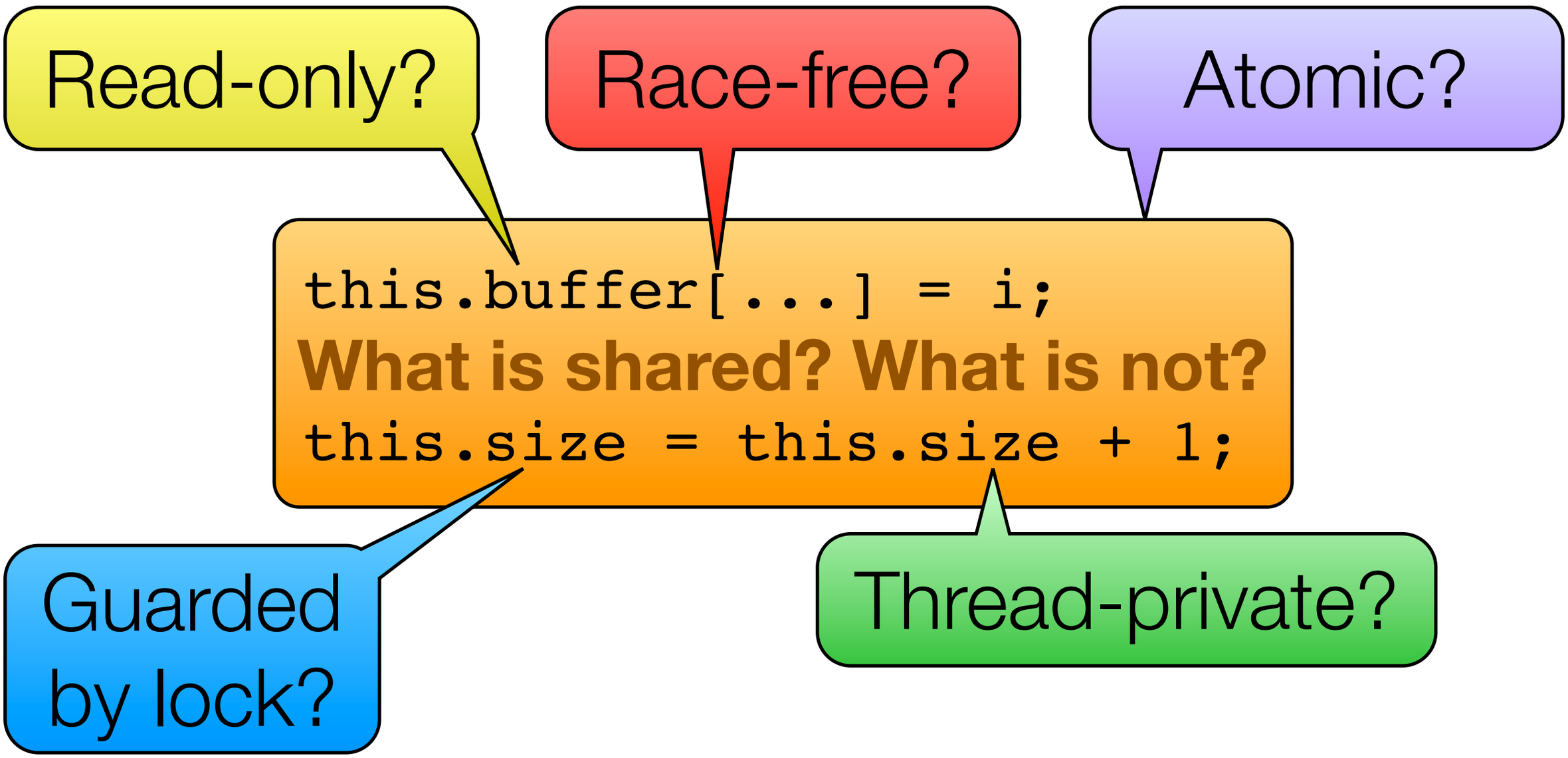
Guarded by lock?

Thread-private?

Implicitly Shared Memory



Implicitly Shared Memory



These are properties of **data** or **isolation**.

Read-only?

Race-free?

Atomic?

```
this.buffer[...] = i;
```

What is shared? What is not?

```
this.size = this.size + 1;
```

Guarded by lock?

Thread-private?

Data- and Isolation-Centric Analyses

Race detection

e.g. FastTrack [PLDI'09], Goldilocks [PLDI'07], Effective Static Race Detection [PLDI'06]

Sharing specifications

e.g. SharC [PLDI'08], Shoal [PLDI'09], Ownership Policies [POPL'10]

Atomicity violation detection

e.g. Velodrome [PLDI'08], A Type and Effect System for Atomicity [PLDI'03]

Data- and Isolation-Centric Analyses

Race detection

e.g. FastTrack [PLDI'09], Goldilocks [PLDI'07], Effective Static Race Detection [PLDI'06]

Are all accesses to **location x** well-synchronized?

Sharing specifications

e.g. SharC [PLDI'08], Shoal [PLDI'09], Ownership Policies [POPL'10]

Atomicity violation detection

e.g. Velodrome [PLDI'08], A Type and Effect System for Atomicity [PLDI'03]

Data- and Isolation-Centric Analyses

Race detection

e.g. FastTrack [PLDI'09], Goldilocks [PLDI'07], Effective Static Race Detection [PLDI'06]

Are all accesses to **location x** well-synchronized?

Sharing specifications

e.g. SharC [PLDI'08], Shoal [PLDI'09], Ownership Policies [POPL'10]

Which **locations** may be shared?

Atomicity violation detection

e.g. Velodrome [PLDI'08], A Type and Effect System for Atomicity [PLDI'03]

Data- and Isolation-Centric Analyses

Race detection

e.g. FastTrack [PLDI'09], Goldilocks [PLDI'07], Effective Static Race Detection [PLDI'06]

Are all accesses to **location x** well-synchronized?

Sharing specifications

e.g. SharC [PLDI'08], Shoal [PLDI'09], Ownership Policies [POPL'10]

Which **locations** may be shared?

Atomicity violation detection

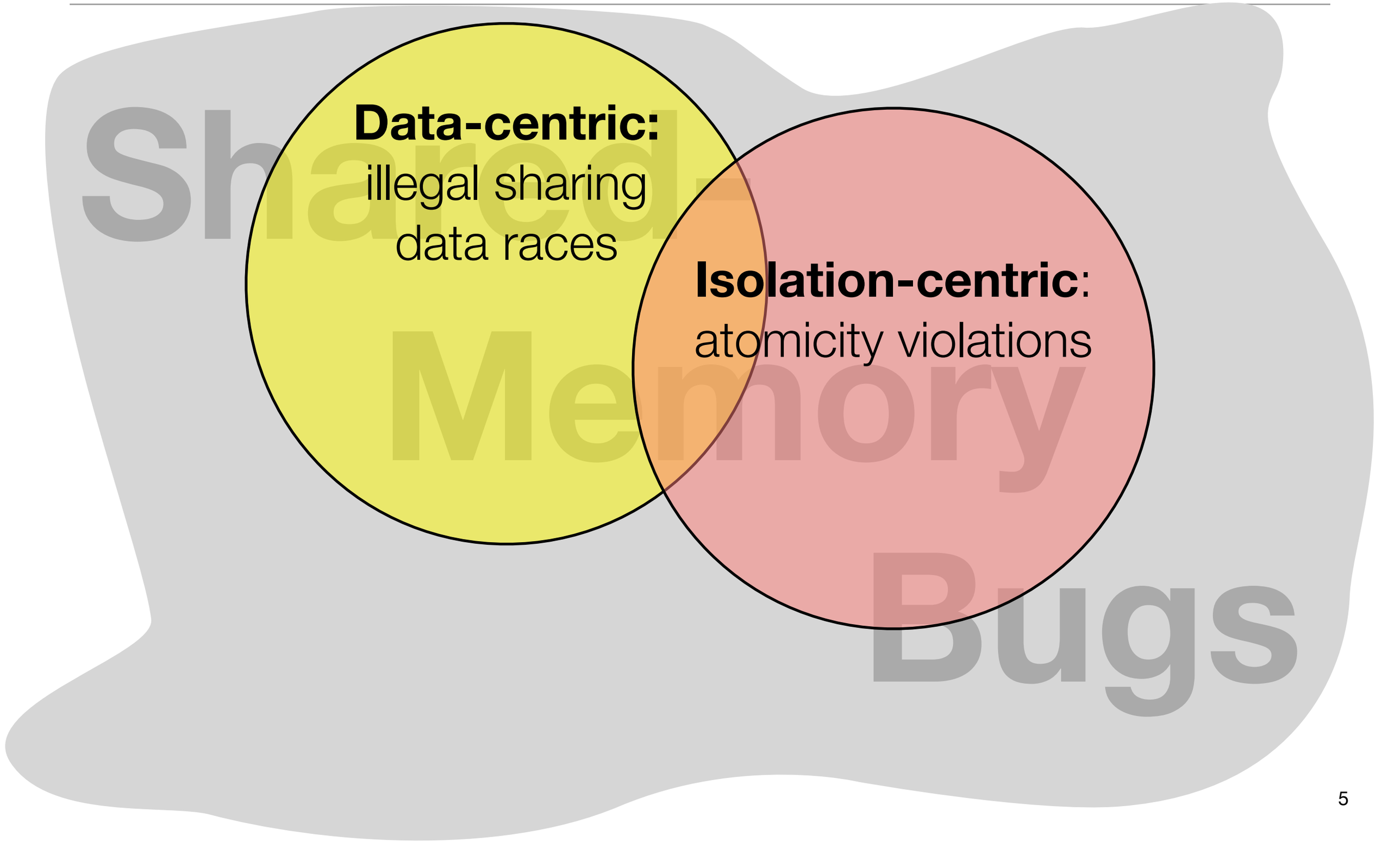
e.g. Velodrome [PLDI'08], A Type and Effect System for Atomicity [PLDI'03]

Are accesses in this code section **isolated**?

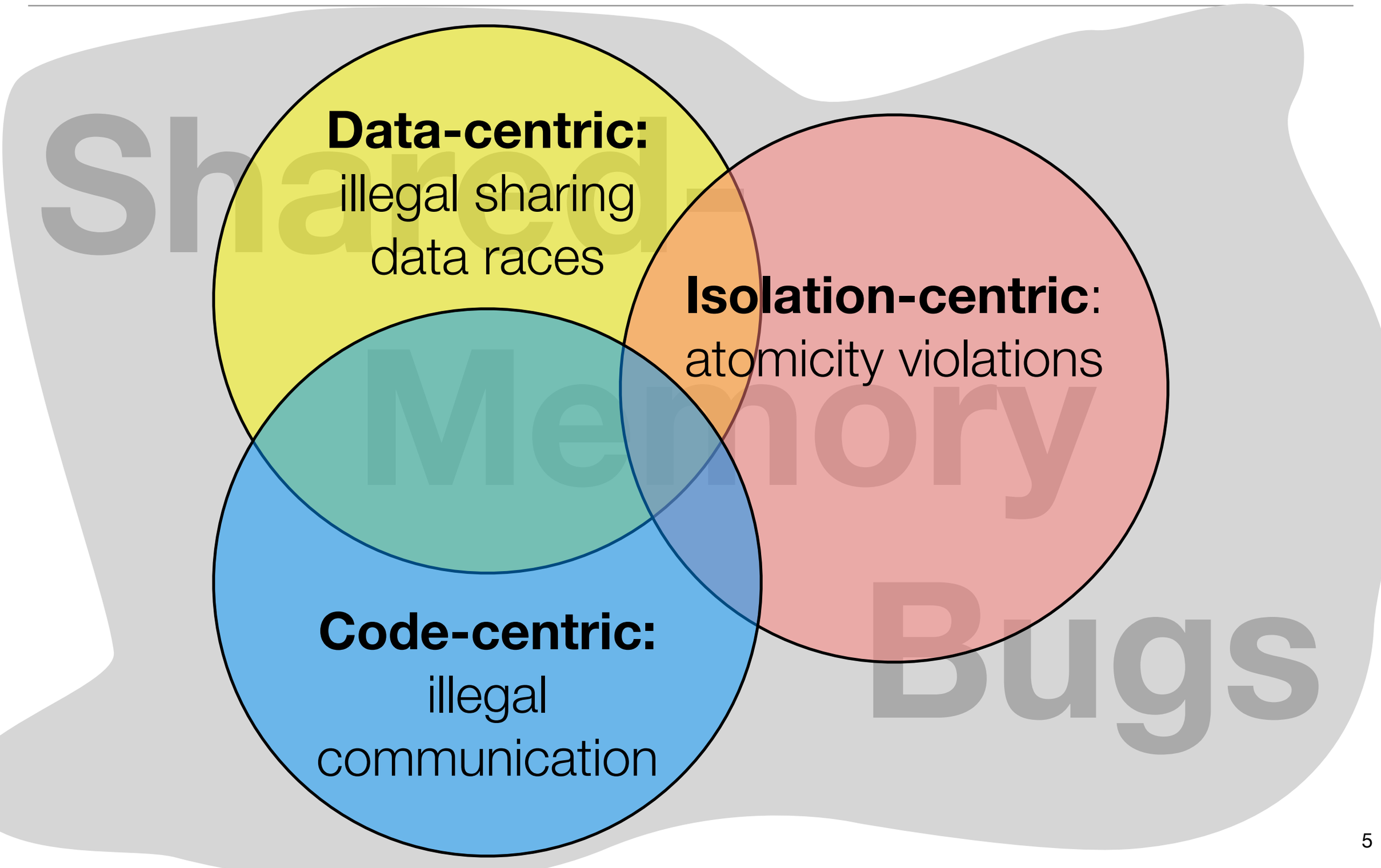
What Shared-Memory Bugs Can We Catch?

Shared-Memory Bugs

What Shared-Memory Bugs Can We Catch?



What Shared-Memory Bugs Can We Catch?



Outline

A Code-Centric View of Shared-Memory

Code-Communication Specification Language

- Making Specifications Modular and Concise
- Specification Language Evaluation

Dynamic Specification Checker

- Making Communication Checking Fast Enough
- Performance Evaluation

Specification Constructs

Specification Constructs

Module A set of related methods
(often aligned with data abstractions)

Specification Constructs

Module	A set of related methods (often aligned with data abstractions)
Module Specification	Which pairs of methods may communicate

Specification Constructs

Module	A set of related methods (often aligned with data abstractions)
Module Specification	Which pairs of methods may communicate
Module Interface	Which communication is encapsulated or visible to callers outside the module

Specification Constructs

Module	A set of related methods (often aligned with data abstractions)
Module Specification	Which pairs of methods may communicate
Module Interface	Which communication is encapsulated or visible to callers outside the module
Inlining	Assigns communication to the caller

Inter-Thread Communication

Writer Thread

```
buffer[3] = ...;
```

Inter-Thread Communication

Writer Thread | Reader Thread

```
buffer[3] = ...;    return buffer[3];
```

communication

Inter-Thread Communication

Writer Thread | Reader Thread

in enqueue(...): → in dequeue(...):

```
buffer[3] = ...;    return buffer[3];
```

communication

Inter-Thread Communication

Writer Thread	Reader Thread
in enqueue(...):	in dequeue(...):
`buffer[3] = ...;`	`return buffer[3];`

Code communication is **directed**.

Inter-Thread Communication

Writer Thread → Reader Thread

in produce(...): → in consume(...):

in enqueue(...): → in dequeue(...):

```
buffer[3] = ...;    return buffer[3];
```

communication

Inter-Thread Communication

Writer Thread → Reader Thread

in produce(...): → in consume(...):

in enqueue(...): → in dequeue(...):

```
buffer[3] = ...;      return buffer[3];
```

Code communication is **layered**.

Communication Modules

```
package pipeline;
import buffer.BoundedBuffer;
class Pipeline {
  BoundedBuffer pipe;

  // Producer threads
  void produce() {
    ... pipe.enqueue(...); ...
  }

  // Consumer threads
  void consume() {
    ... = pipe.dequeue(); ...
  }
}
```

```
package buffer;
public class BoundedBuffer {
  Item[] buffer = new Item[10];
  int size = 0;

  public synchronized void enqueue(Item i) {
      while (size == buffer.length) wait();
      buffer[...] = i;
      size++; ...
      notifyAll();
  }

  public synchronized Item dequeue() {
      while (size == 0) wait();
      size--; ...
      notifyAll();
      return buffer[...];
  }
}
```

Communication Modules

```
package pipeline;
import buffer.BoundedBuffer;
class Pipeline {
  BoundedBuffer pipe;

  // Producer threads
  void produce() {
    ... pipe.enqueue(...); ...
  }

  // Consumer threads
  void consume() {
    ... = pipe.dequeue(); ...
  }
}
```

Module Specification

```
package buffer;
public class BoundedBuffer {
  Item[] buffer = new Item[10];
  int size = 0;

  public synchronized void enqueue(Item i) {
      while (size == buffer.length) wait();
      buffer[...] = i;
      size++; ...
      notifyAll();
  }

  public synchronized Item dequeue() {
      while (size == 0) wait();
      size--; ...
      notifyAll();
      return buffer[...];
  }
}
```

Communication Modules

Module Specification

```
package pipeline;
import buffer.BoundedBuffer;
class Pipeline {
  BoundedBuffer pipe;

  // Producer threads
  void produce() {
    ... pipe.enqueue(...); ...
  }

  // Consumer threads
  void consume() {
    ... = pipe.dequeue(); ...
  }
}
```

```
package buffer;
public class BoundedBuffer {
  Item[] buffer = new Item[10];
  int size = 0;

  public synchronized void enqueue(Item i) {
      while (size == buffer.length) wait();
      buffer[...] = i;
      size++; ...
      notifyAll();
  }

  public synchronized Item dequeue() {
      while (size == 0) wait();
      size--; ...
      notifyAll();
      return buffer[...];
  }
}
```

Communication Modules

Module Specification

```
package pipeline;
import buffer.BoundedBuffer;
class Pipeline {
  BoundedBuffer pipe;

  // Producer threads
  void produce() {
    ... pipe.enqueue(...); ...
  }

  // Consumer threads
  void consume() {
    ... = pipe.dequeue(); ...
  }
}
```

```
package buffer;
public class BoundedBuffer {
  Item[] buffer = new Item[10];
  int size = 0;

  public synchronized void enqueue(Item i) {
      while (size == buffer.length) wait();
      buffer[...] = i;
      size++; ...
      notifyAll();
  }

  public synchronized Item dequeue() {
      while (size == 0) wait();
      size--; ...
      notifyAll();
      return buffer[...];
  }
}
```

Communication Modules

```
package pipeline;
import buffer.BoundedBuffer;
class Pipeline {
  BoundedBuffer pipe;

  // Producer threads
  void produce() {
    ... pipe.enqueue(...); ...
  }

  // Consumer threads
  void consume() {
    ... = pipe.dequeue(); ...
  }
}
```

Module Specification

```
package buffer;
public class BoundedBuffer {
  Item[] buffer = new Item[10];
  int size = 0;

  public synchronized void enqueue(Item i) {
      while (size == buffer.length) wait();
      buffer[...] = i;
      size++; ...
      notifyAll();
  }

  public synchronized Item dequeue() {
      while (size == 0) wait();
      size--; ...
      notifyAll();
      return buffer[...];
  }
}
```

Communication Modules

```
package pipeline;
import buffer.BoundedBuffer;
class Pipeline {
  BoundedBuffer pipe;

  // Producer threads
  void produce() {
    ... pipe.enqueue(...); ...
  }

  // Consumer threads
  void consume() {
    ... = pipe.dequeue(); ...
  }
}
```

Module Specification

```
package buffer;
public class BoundedBuffer {
  Item[] buffer = new Item[10];
  int size = 0;

  public synchronized void enqueue(Item i) {
      while (size == buffer.length) wait();
      buffer[...] = i;
      size++; ...
      notifyAll();
  }

  public synchronized Item dequeue() {
      while (size == 0) wait();
      size--; ...
      notifyAll();
      return buffer[...];
  }
}
```

Checking Communication Specifications

Writer Thread

in produce(...):

```
buffer[3] = ...;
```

Reader Thread

in consume(...):

```
return buffer[3];
```

in enqueue(...):

in dequeue(...):

Checking Communication Specifications

Writer Thread

Reader Thread

in produce(...):

in enqueue(...):

in consume(...):

in dequeue(...):

```
buffer[3] = ...;
```

```
return buffer[3];
```

Checking Communication Specifications

Writer Thread | Reader Thread

in produce(...): | in consume(...):

in enqueue(...): | in dequeue(...):

```
buffer[3] = ...;    return buffer[3];
```

Checking Communication Specifications

Writer Thread	Reader Thread
in produce(...):	in consume(...):

in enqueue(...): ✔ in dequeue(...):

```
buffer[3] = ...;    return buffer[3];
```

Checking Communication Specifications

Writer Thread | Reader Thread

in produce(...): ✔ in consume(...):

in enqueue(...): ✔ in dequeue(...):

```
buffer[3] = ...;    return buffer[3];
```

Checking Communication Specifications

Writer Thread ✔ Reader Thread

in produce(...): ✔→ in consume(...):

in enqueue(...): ✔→ in dequeue(...):

```
buffer[3] = ...;    return buffer[3];
```

Communication Module Interfaces

Writer Thread

Reader Thread

in consume(...):

in produce(...):

in dequeue(...):

in enqueue(...):

```
size--;
```

```
... = size;
```

Communication Module Interfaces

Writer Thread | Reader Thread

in consume(...): | in produce(...):

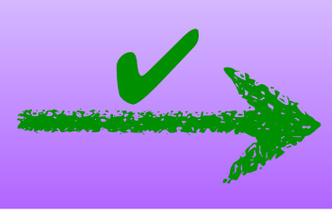
in dequeue(...): ✔ in enqueue(...):

```
size--;          ... = size;
```

Communication Module Interfaces

Writer Thread | Reader Thread

in consume(...):  in produce(...):

in dequeue(...):  in enqueue(...):

```
size--;          ... = size;
```

Communication Module Interfaces

Module Specification

```
package pipeline;
import buffer.BoundedBuffer;
class Pipeline {
  BoundedBuffer pipe;

  // Producer threads
  void produce() {
    ...; pipe.enqueue(...); ...
  }

  // Consumer threads
  void consume() {
    ... = pipe.dequeue(); ...
  }
}
```

```
package buffer;
public class BoundedBuffer {
  Item[] buffer = new Item[10];
  int size = 0;

  public synchronized void enqueue(Item i) {
      while (size == buffer.length) wait();
      buffer[...] = i;
      size++; ...
      notifyAll();
  }

  public synchronized Item dequeue() {
      while (size == 0) wait();
      size--; ...
      notifyAll();
      return buffer[...];
  }
}
```

Communication Module Interfaces

Module Specification

```
package pipeline;
import buffer.BoundedBuffer;
class Pipeline {
  BoundedBuffer pipe;

  // Producer threads
  void produce() {
    ...; pipe.enqueue(...); ...
  }

  // Consumer threads
  void consume() {
    ... = pipe.dequeue(); ...
  }
}
```

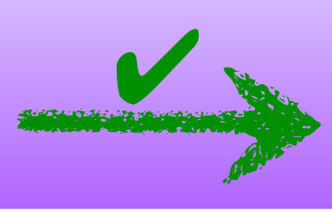
```
package buffer;
public class BoundedBuffer {
  Item[] buffer = new Item[10];
  int size = 0;

  public synchronized void enqueue(Item i) {
    while (size == buffer.length) wait();
    buffer[...] = i;
    size++; ...
    notifyAll();
  }

  public synchronized Item dequeue() {
    while (size == 0) wait();
    size--; ...
    notifyAll();
    return buffer[...];
  }
}
```

Module Interface

Communication Module Interfaces

Writer Thread Reader Thread

in consume(...):  in produce(...):

in dequeue(...):  in enqueue(...):

```
size--;          ... = size;
```

Communication Module Interfaces

Writer Thread	Reader Thread
in consume(...):	in produce(...):

in dequeue(...): ✔ in enqueue(...):

```
size--;          ... = size;
```

Communication Module Interfaces

Writer Thread	Reader Thread
in consume(...):	in produce(...):
in dequeue(...): ✔	in enqueue(...):
`size--;`	`... = size;`

Communication Module Interfaces

Writer Thread ✔ Reader Thread

in consume(...): in produce(...):

encapsulated

in dequeue(...): ✔ in enqueue(...):

```
size--;          ... = size;
```

Communication Inlining

Writer Thread

Reader Thread

in enqueue(...):

in arrayCopy(...):

in dequeue(...):

```
buffer[3] = ...;     return buffer[3];
```

Communication Inlining

Writer Thread

Reader Thread

in enqueue(...):

in arrayCopy(...):

in dequeue(...):

```
buffer[3] = ...;      return buffer[3];
```

arrayCopy communicates only **for its caller**.

Communication Inlining

Writer Thread

Reader Thread

in enqueue(...):

@Inline arrayCopy(...):

in dequeue(...):

```
buffer[3] = ...;
```

```
return buffer[3];
```

arrayCopy communicates only **for its caller**.

Communication Inlining

Writer Thread	Reader Thread
in enqueue(...):	in dequeue(...):
`buffer[3] = ...;`	`return buffer[3];`

arrayCopy communicates only **for its caller**.

Communication Inlining

Writer Thread ✔ Reader Thread

in enqueue(...): ✔ in dequeue(...):

```
buffer[3] = ...;     return buffer[3];
```

arrayCopy communicates only **for its caller**.

Specification Constructs

Specification Constructs

Module A set of related methods
(often aligned with data abstractions)

Specification Constructs

Module	A set of related methods (often aligned with data abstractions)
Module Specification	Which pairs of methods may communicate

Specification Constructs

Module	A set of related methods (often aligned with data abstractions)
Module Specification	Which pairs of methods may communicate
Module Interface	Which communication is encapsulated or visible to callers outside the module

Specification Constructs

Module	A set of related methods (often aligned with data abstractions)
Module Specification	Which pairs of methods may communicate
Module Interface	Which communication is encapsulated or visible to callers outside the module
Inlining	Assigns communication to the caller

Evaluation: Specification Size

DaCapo

Java Grande

Evaluation: Specification Size

	Benchmark	LOC
DaCapo	Avrora	70,000
	Batik	190,000
	Xalan	180,000
Java Grande	Crypt	300
	LUFact	500
	MolDyn	500
	MonteCarlo	1,200
	RayTracer	700
	Series	200
	SOR	200
	Sparsematmult	200

Evaluation: Specification Size

	Benchmark	LOC	Total Annotations	**Ann. / KLOC**
DaCapo	Avrora	70,000	175	**2.5**
	Batik	190,000	16	**0.01**
	Xalan	180,000	90	**0.5**
Java Grande	Crypt	300	16	53
	LUFact	500	15	30
	MolDyn	500	39	78
	MonteCarlo	1,200	19	16
	RayTracer	700	37	53
	Series	200	10	50
	SOR	200	14	70
	Sparsematmult	200	9	45

Evaluation: Specification Size

	Benchmark	LOC	Total Annotations	**Ann. / KLOC**	Methods	Methods Annotated	**% Methods Annotated**
DaCapo	Avrora	70,000	175	**2.5**	9,775	85	**0.9%**
	Batik	190,000	16	**0.01**	15,547	8	**0.05%**
	Xalan	180,000	90	**0.5**	7,854	42	**0.5%**
Java Grande	Crypt	300	16	53	17	5	29%
	LUFact	500	15	30	29	6	21%
	MolDyn	500	39	78	27	16	59%
	MonteCarlo	1,200	19	16	172	11	6%
	RayTracer	700	37	53	77	15	19%
	Series	200	10	50	15	6	40%
	SOR	200	14	70	13	5	38%
	Sparsematmult	200	9	45	12	4	33%

Specification Expressiveness

Specification Expressiveness

Strengths:

- ✓Concise and intuitive
- ✓Encapsulation useful in many benchmarks
- ✓Sensitive to error

Specification Expressiveness

Strengths:

- ✓Concise and intuitive
- ✓Encapsulation useful in many benchmarks
- ✓Sensitive to error

Limitations / Future Work:

- Improve support for non-layered communication
- Integrate data-centric properties to reduce specification size

Specification Expressiveness

Strengths:

- ✓Concise and intuitive
- ✓Encapsulation useful in many benchmarks
- ✓Sensitive to error

Limitations / Future Work:

- Improve support for non-layered communication
- Integrate data-centric properties to reduce specification size

Also in the Paper:

- Java annotation syntax
- Formal semantics

Outline

A Code-Centric View of Shared-Memory

Code-Communication Specification Language

- Making Specifications Modular and Concise
- Specification Language Evaluation

Dynamic Specification Checker

- Making Communication Checking Fast Enough
- Performance Evaluation

Fundamental Instrumentation Costs

```
class C {
  int x;
  State x__lastWriter;
```

Fundamental Instrumentation Costs

```
class C {
  int x;
  State x__lastWriter;
```

write Store current thread and call stack as last writer.

Fundamental Instrumentation Costs

```
class C {
  int x;
  State x__lastWriter;
```

write	Store current thread and call stack as last writer.
read	**Check** if communication is allowed from last writer to current reader.

Optimizing Read Checks

Check		Action	Mem. Ops.
Same thread?	✔		**1**
Full check passes?	✔		**>30**
Else illegal.	✘	Throw exception.	

Optimizing Read Checks

Check		Action	Mem. Ops.
Same thread?	✔		**1**
Full check passes?	✔		**>30**
Else illegal.	✘	Throw exception.	

Optimizing Read Checks

Check		Action	Mem. Ops.
Same thread?	✔		**1**
Full check passes?	✔	**Add pair to global memo table.**	**>30**
Else illegal.	✘	Throw exception.	

Optimizing Read Checks

Check		Action	Mem. Ops.
Same thread?	✔		**1**
Stack pair in global memo table?	✔		**12**
Full check passes?	✔	Add pair to global memo table.	**>30**
Else illegal.	✘	Throw exception.	

Optimizing Read Checks

Check		Action	Mem. Ops.
Same thread?	✔		**1**
Stack pair in global memo table?	✔	**Add writer stack ID to reader stack's cache.**	**12**
Full check passes?	✔	Add pair to global memo table.	**>30**
Else illegal.	✘	Throw exception.	

Optimizing Read Checks

Check		Action	Mem. Ops.
Same thread?	✔		**1**
Writer stack ID in reader stack's cache?	✔		**4**
Stack pair in global memo table?	✔	Add writer stack ID to reader stack's cache.	**12**
Full check passes?	✔	Add pair to global memo table.	**>30**
Else illegal.	✘	Throw exception.	

Optimizing Read Checks

Check		Action	Mem. Ops.
Same thread?	✔		**1**
Writer stack ID in reader stack's cache?	✔		**4**
Stack pair in global memo table?	✔	Add writer stack ID to reader stack's cache.	**12**
Full check passes?	✔	Add pair to global memo table.	**>30**
Else illegal.	✘	Throw exception.	

Experimental Configuration

Benchmarks	8 Java Grande, large inputs, 8 threads 3 DaCapo 9.12, default inputs, 8 threads
Machine	8-core 2.8GHz Intel Xeon, 10GB RAM Ubuntu 8.10
JVM	HotSpot 64-bit client VM 1.6.0 max heap size 8GB
Data	Average over 10 runs separate performance and profiling

Execution Profile

Execution Profile

> 99.99999% of reads checked on fast paths

Execution Profile

> 99.99999% of reads checked on fast paths

up to 6 billion communicating reads

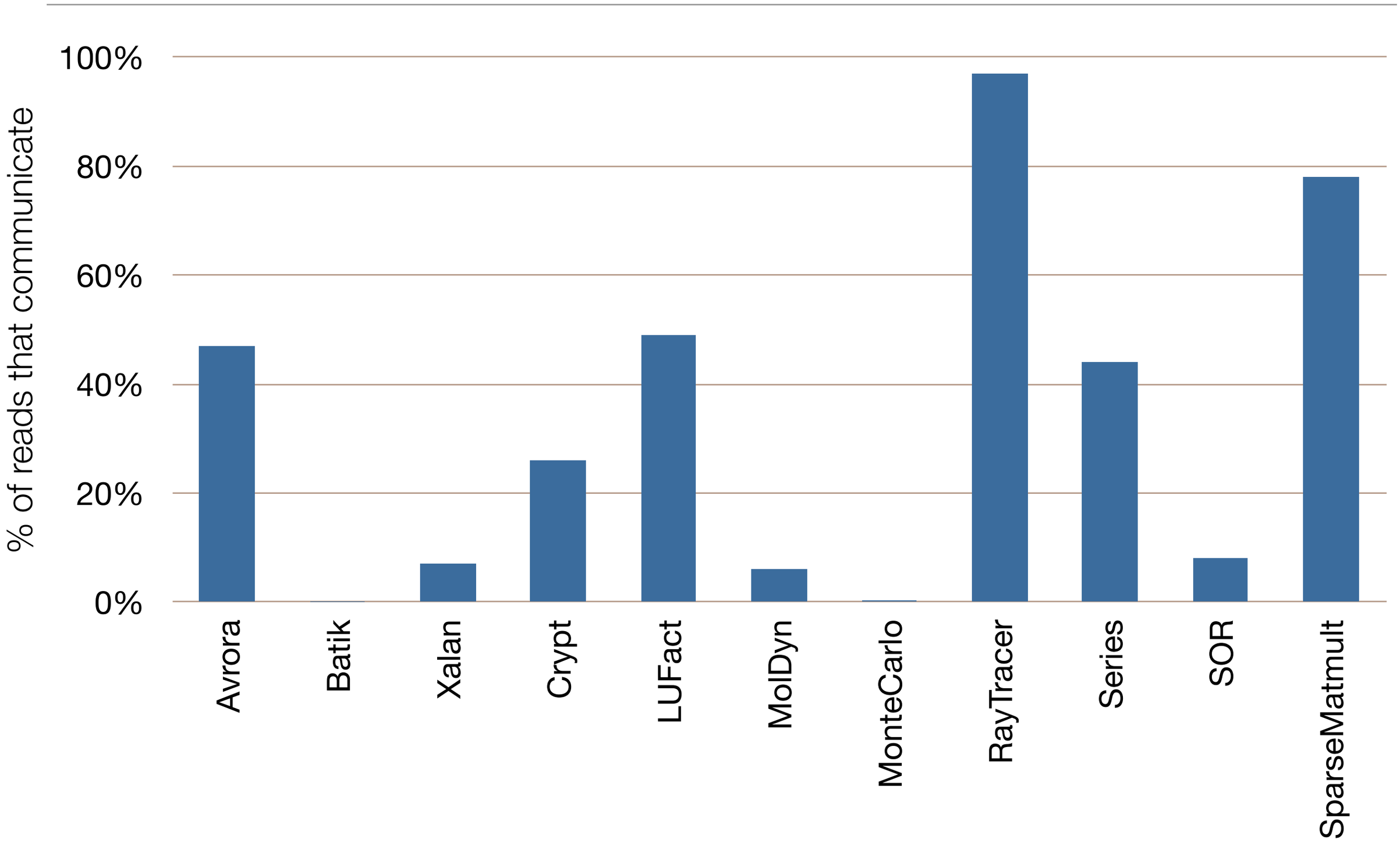
Execution Profile

> 99.99999% of reads checked on fast paths

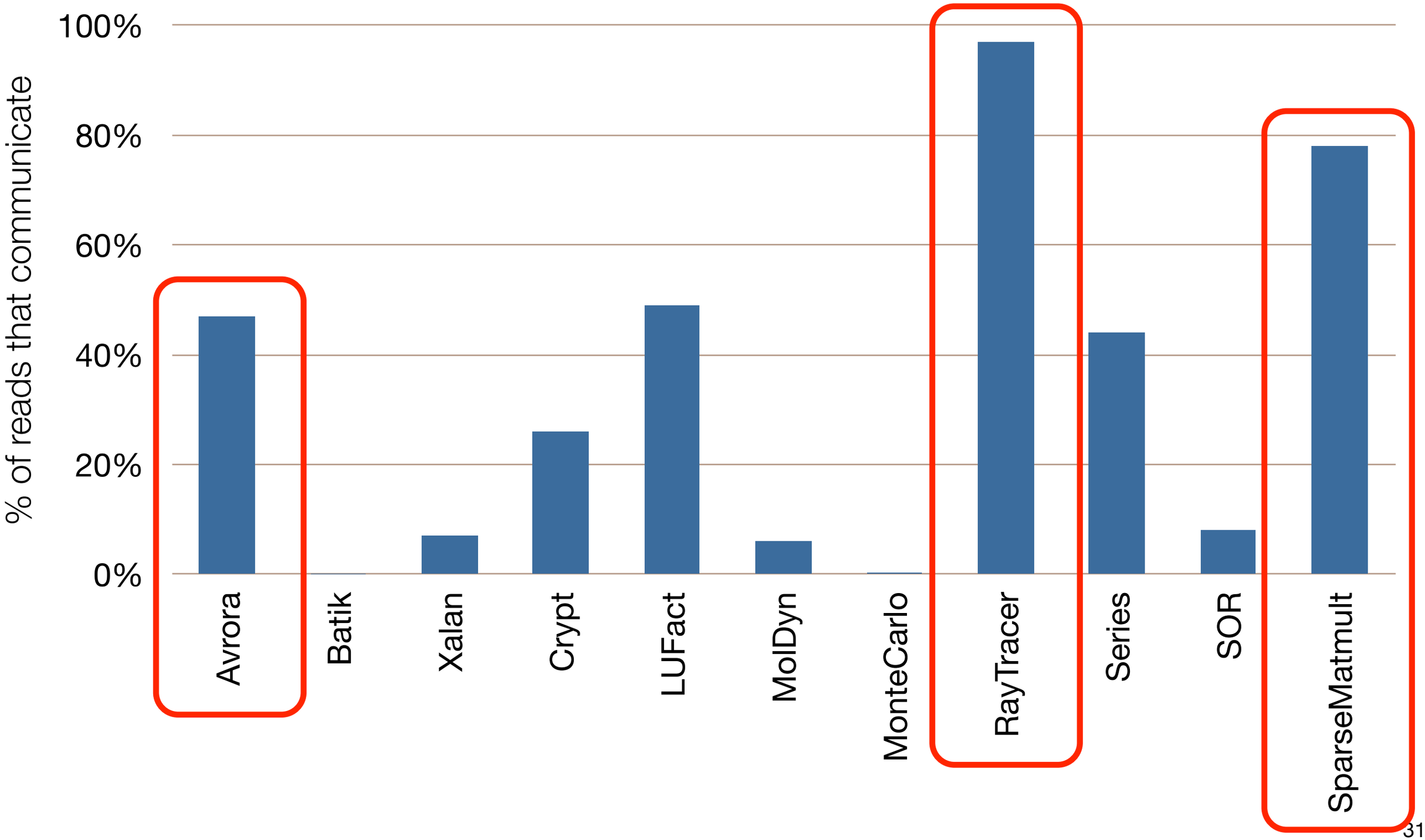
up to 6 billion communicating reads

≤ 697 full stack checks

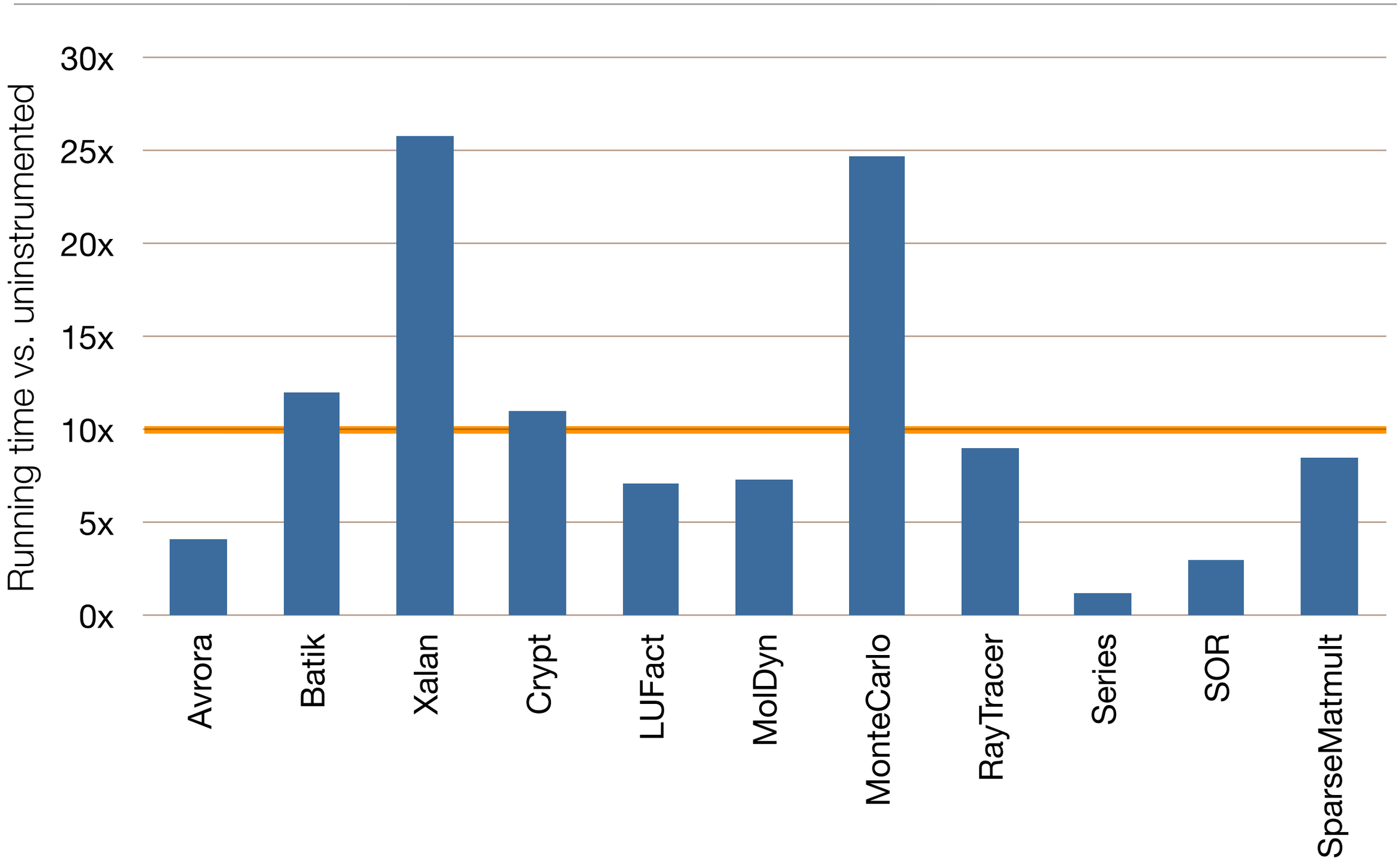
Communicating Read Operations



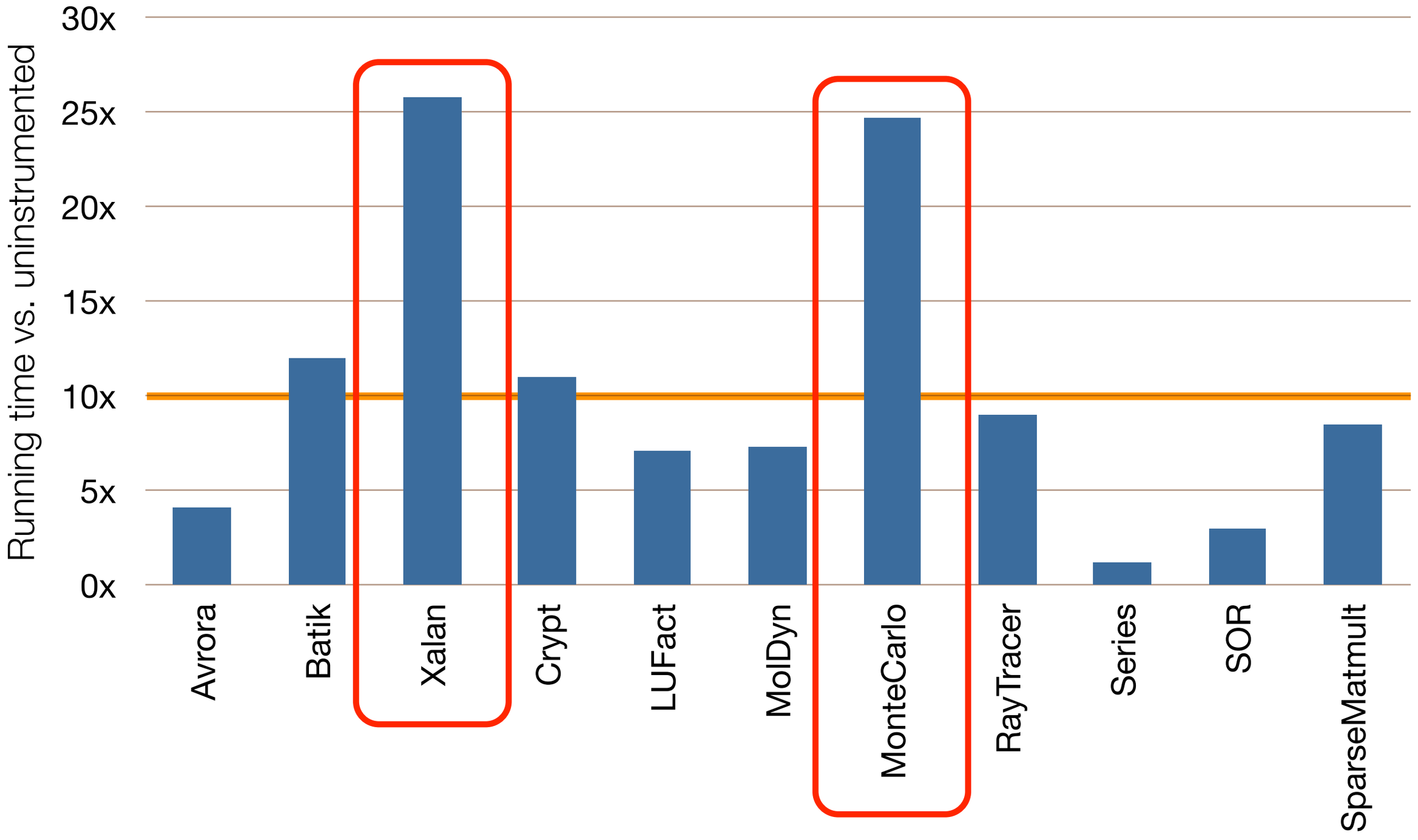
Communicating Read Operations



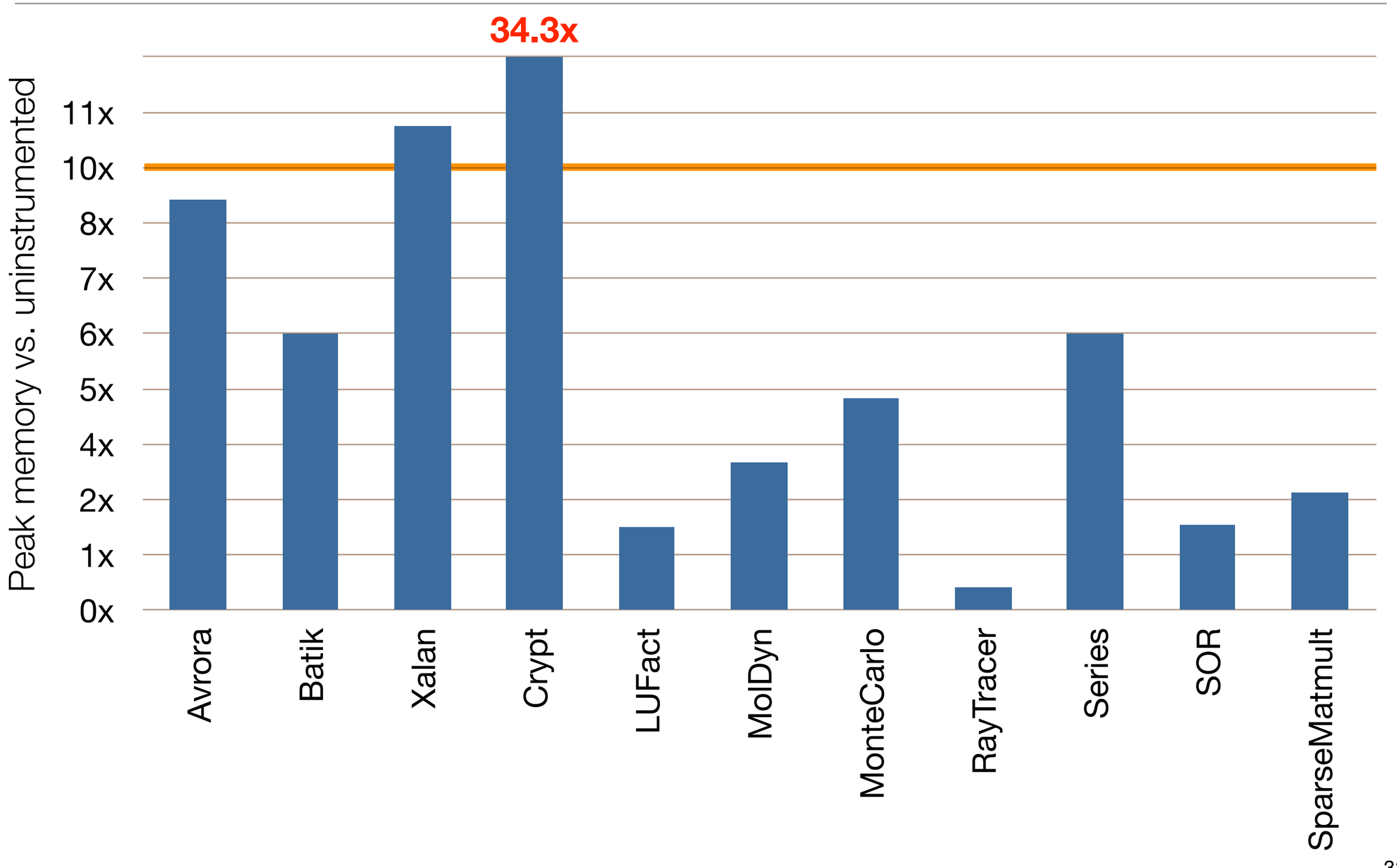
Time Overhead



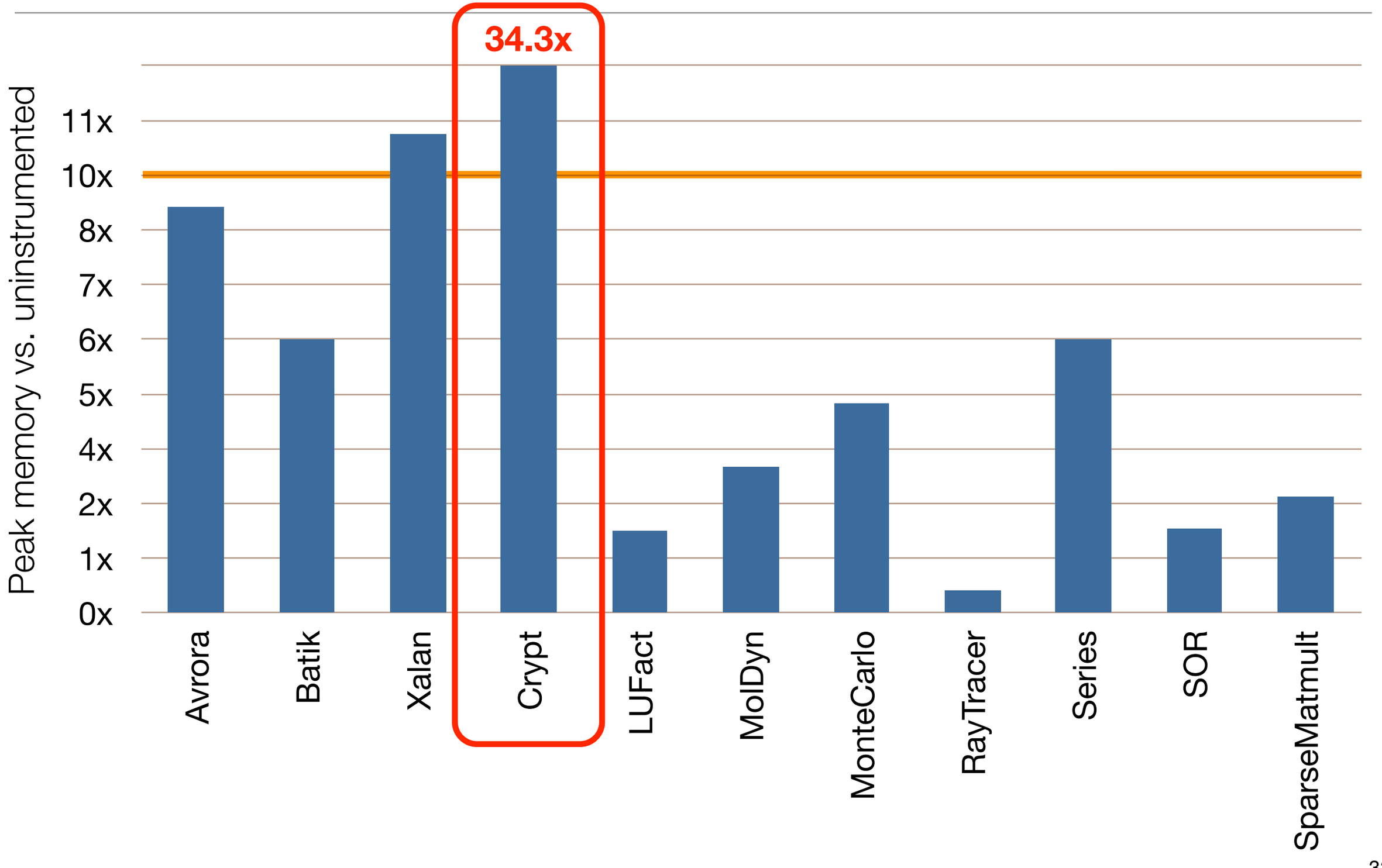
Time Overhead



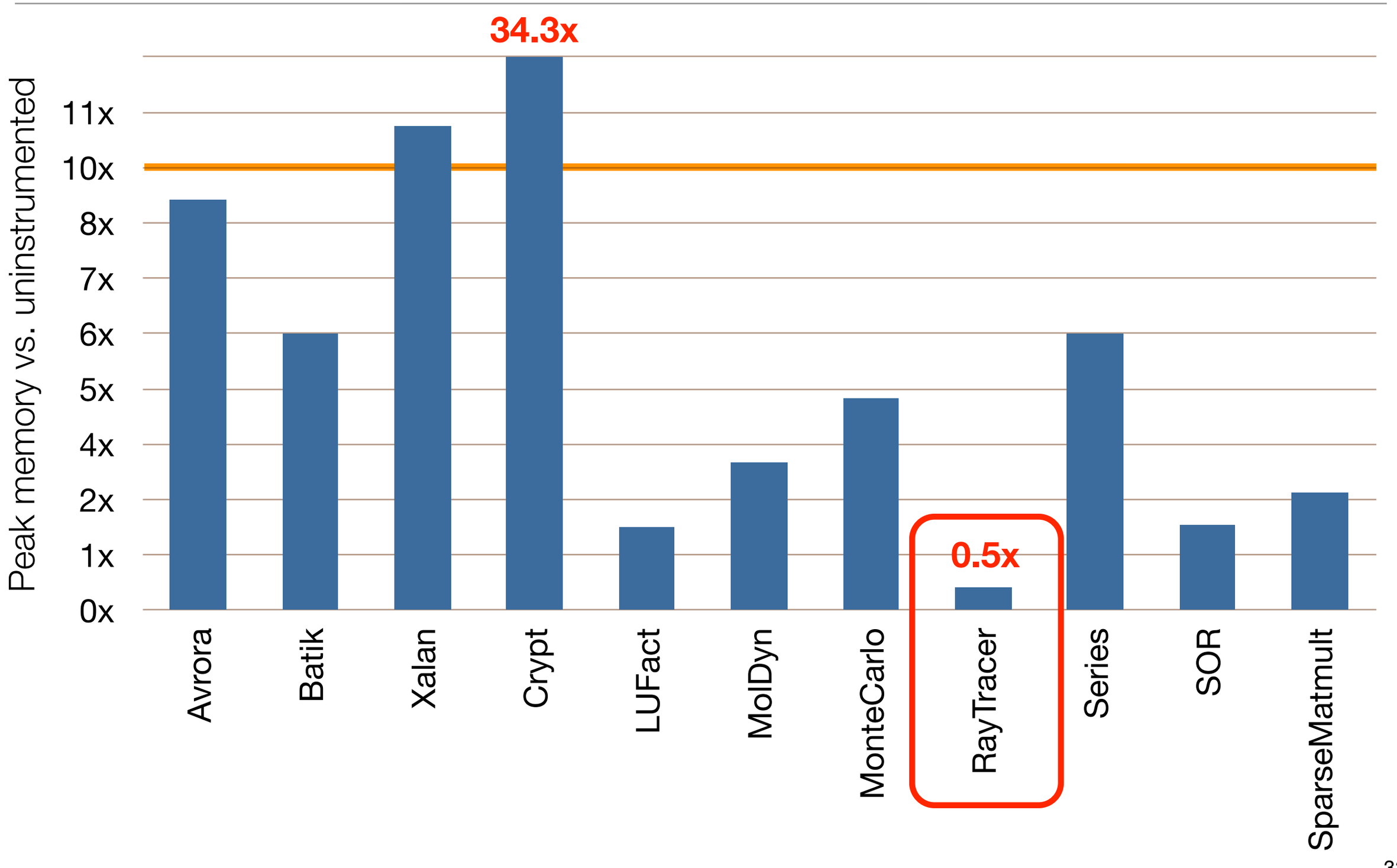
Space Overhead



Space Overhead



Space Overhead



Summary

1. **A Code-Centric View of Shared-Memory**
2. **Code-Communication Specification Language**
   - Concise and modular specifications
   - Fit communication patterns in real programs
3. **Dynamic Specification Checker**
   - Aggressive optimization of communication checks
   - Debugging-level performance

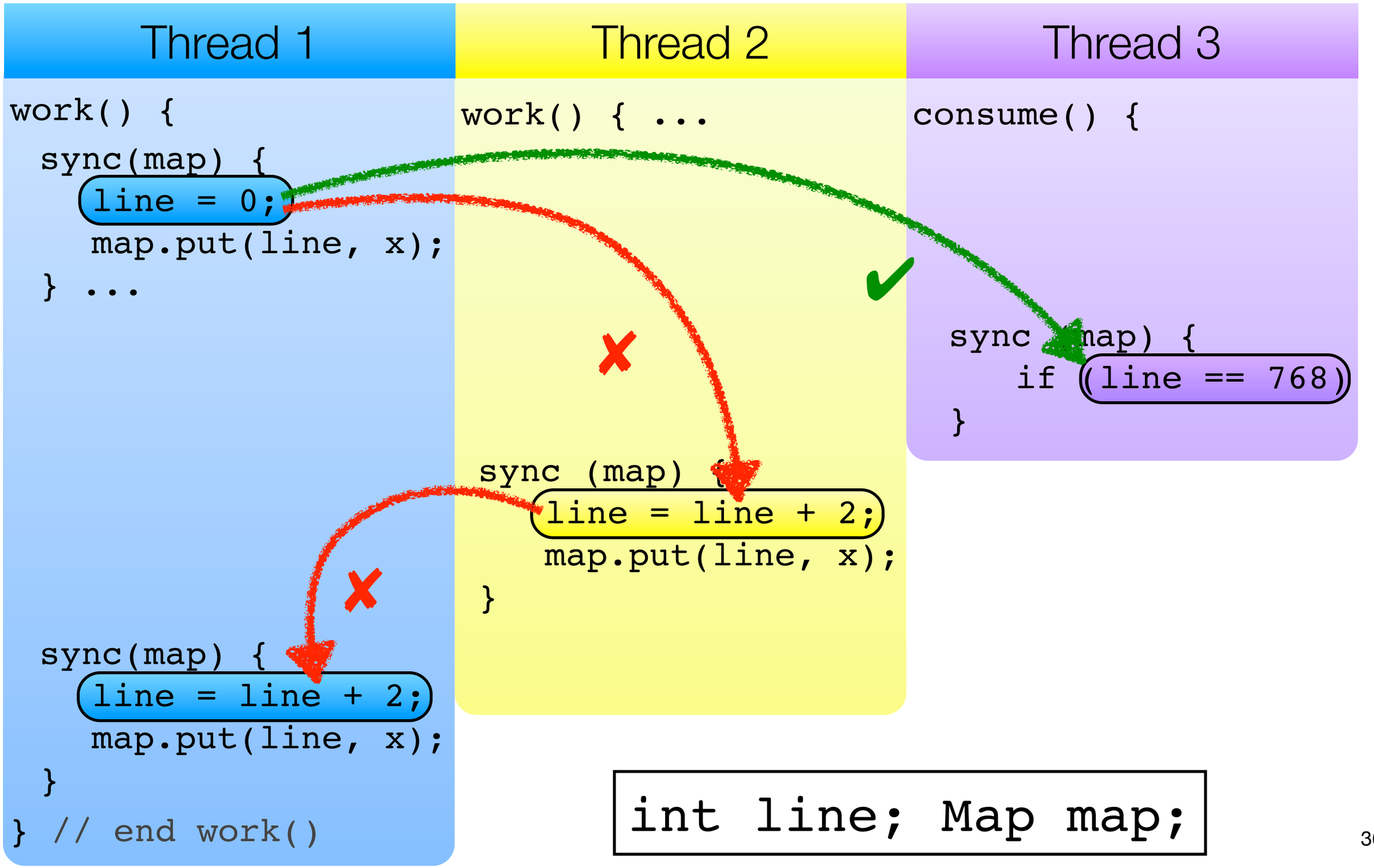
Summary

1. **A Code-Centric View of Shared-Memory**
2. **Code-Communication Specification Language**
   - Concise and modular specifications
   - Fit communication patterns in real programs
3. **Dynamic Specification Checker**
   - Aggressive optimization of communication checks
   - Debugging-level performance

Download: www.cs.washington.edu/homes/bpw/

This slide intentionally not left blank

Communication Specifications, ✔
Race Detection, Sharing Specs, Atomicity Checker



Communication Specifications, ✔
Race Detection, Sharing Specs, Atomicity Checker

Thread 1 | Thread 2 | Thread 3

```
work() {
  sync(map) {
    line = 0;
    map.put(line, x);
  } ...
```

```
work() { ...
    map.put(line, x);
}
```

```
consume() {
  == 768)
```

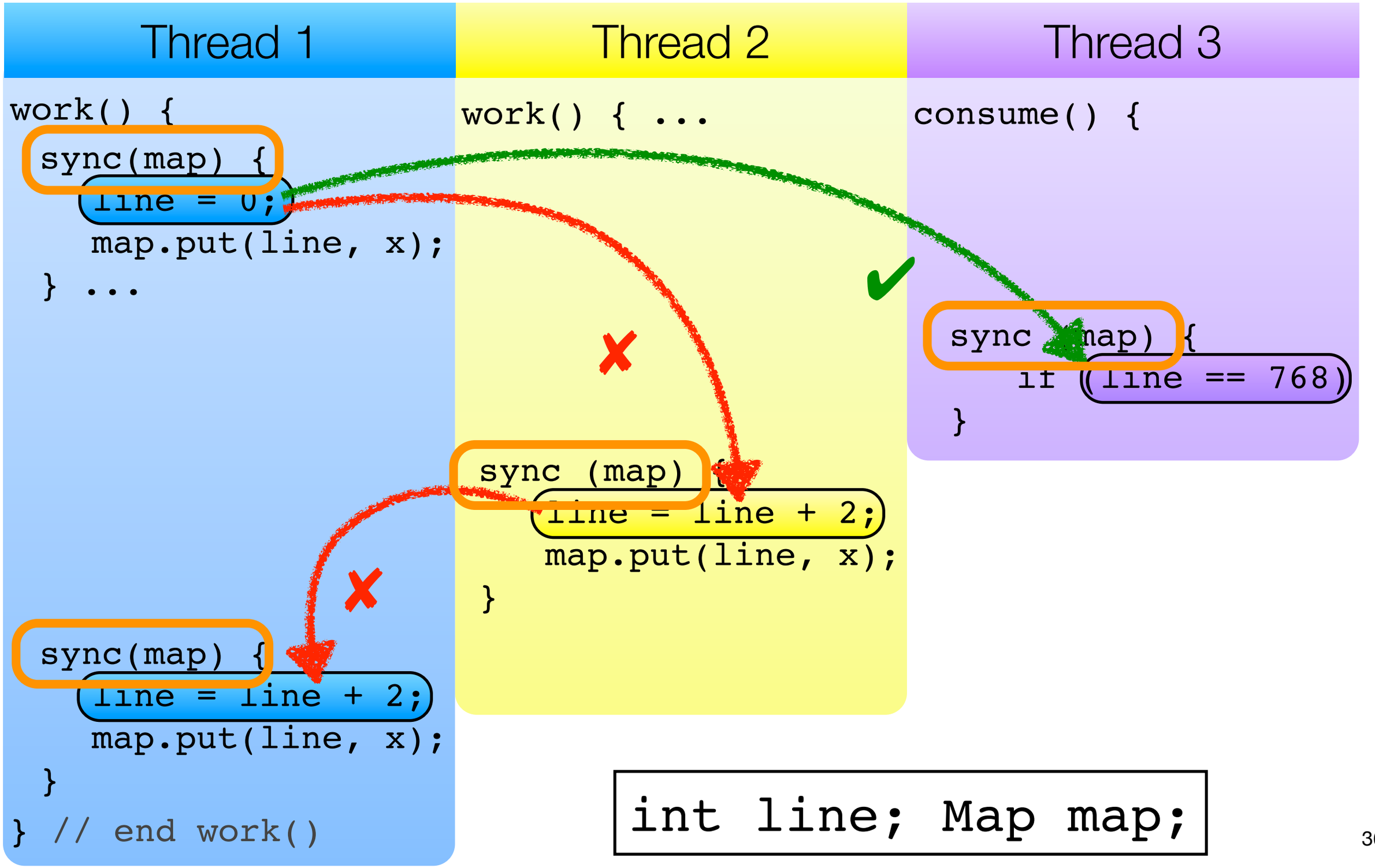
work() ✔ consume()

work() ✘ work()

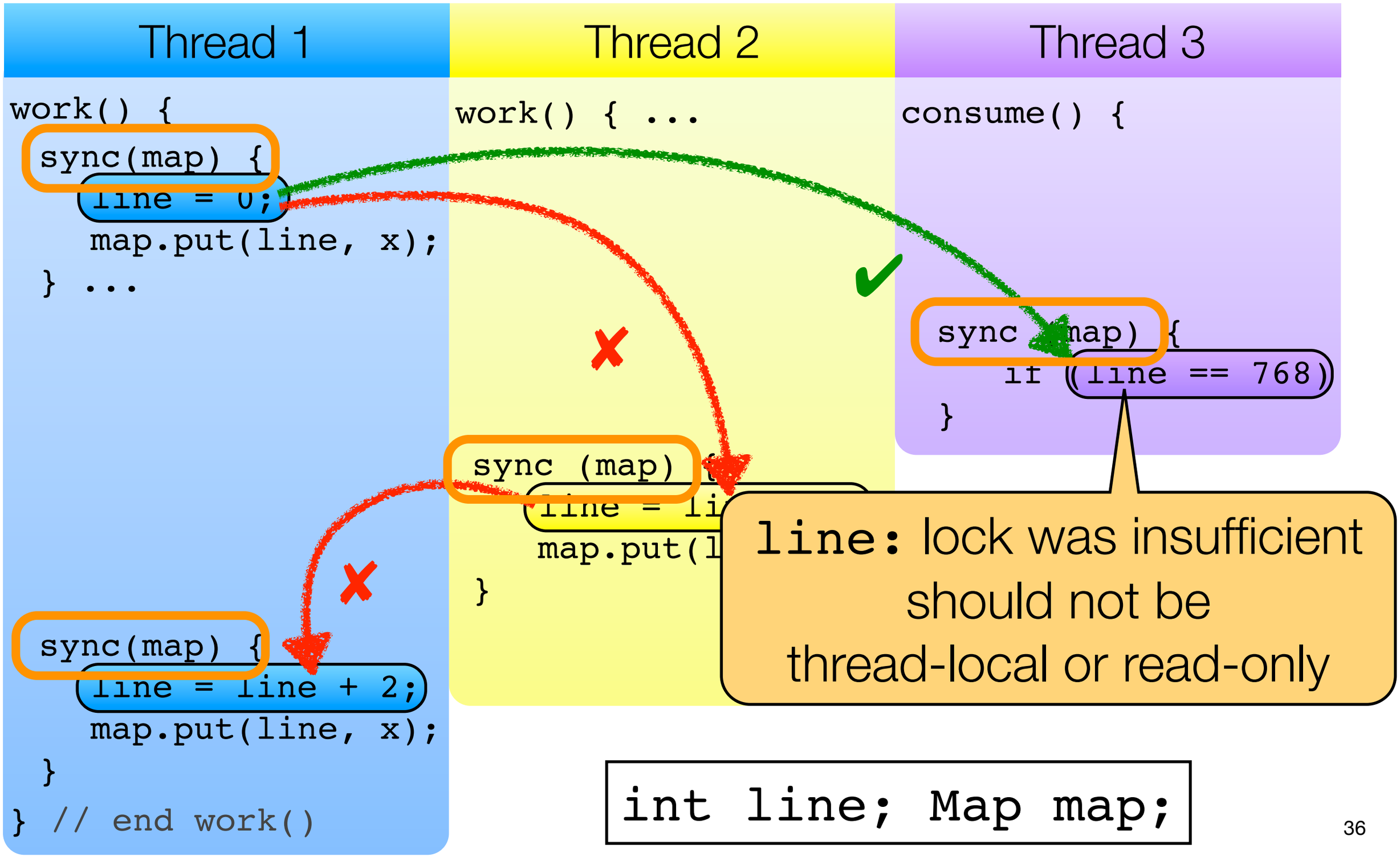
```
  sync(map) {
    line = line + 2;
    map.put(line, x);
  }
} // end work()
```

```
int line; Map map;
```

Communication Specifications, ✔
Race Detection, Sharing Specs, Atomicity Checker



Communication Specifications, ✔
Race Detection ✘, Sharing Specs ✘, Atomicity Checker



Communication Specifications, ✔
Race Detection, Sharing Specs, Atomicity Checker

Thread 1	Thread 2	Thread 3

```
work() {
  sync(map) {
     line = 0;
     map.put(0, x);
  } ...
```

```
work() { ...
```

```
consume() {
```

```
sync (map) {
     line = line + 2;
     map.put(1, x);
}
```

```
  sync(map) {
     line = line + 2;
     map.put(766, x);
  }
} // end work()
```

Correct version is not intended to be atomic.

```
int line; Map map;
```

This slide intentionally not left blank

Callbacks

Writer Thread

Reader Thread

in Simulator.run(...):

in Simulator.run(...):

in EventList.fireAll(...):

in Action.create(...): ✔ in Action.fire(...):

```
buffer[3] = ...;     return buffer[3];
```

Callbacks

Writer Thread

Reader Thread

in Simulator.run(...):

in Simulator.run(...):

in EventList.fireAll(...):

in Action.create(...): ✔ in Action.fire(...):

```
buffer[3] = ...;    return buffer[3];
```

This slide intentionally not left blank