Caching Puts and Gets in a PGAS Language Runtime

Michael Ferguson
Cray Inc.

Daniel Buettner
Laboratory for
Telecommunication Sciences

September 17, 2015

Safe Harbor Statement

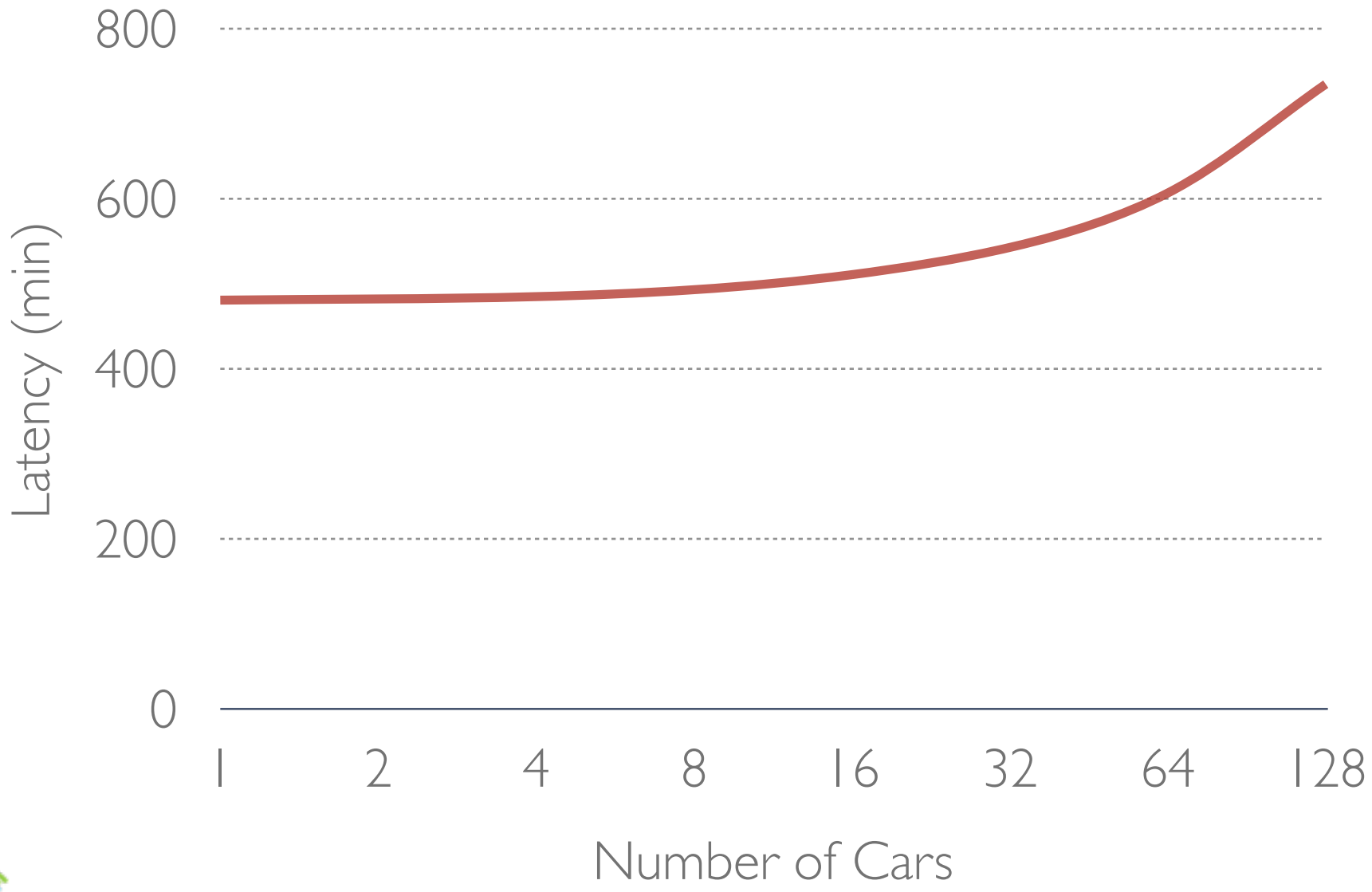
This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

CC Flickr/Daniel Jolivet

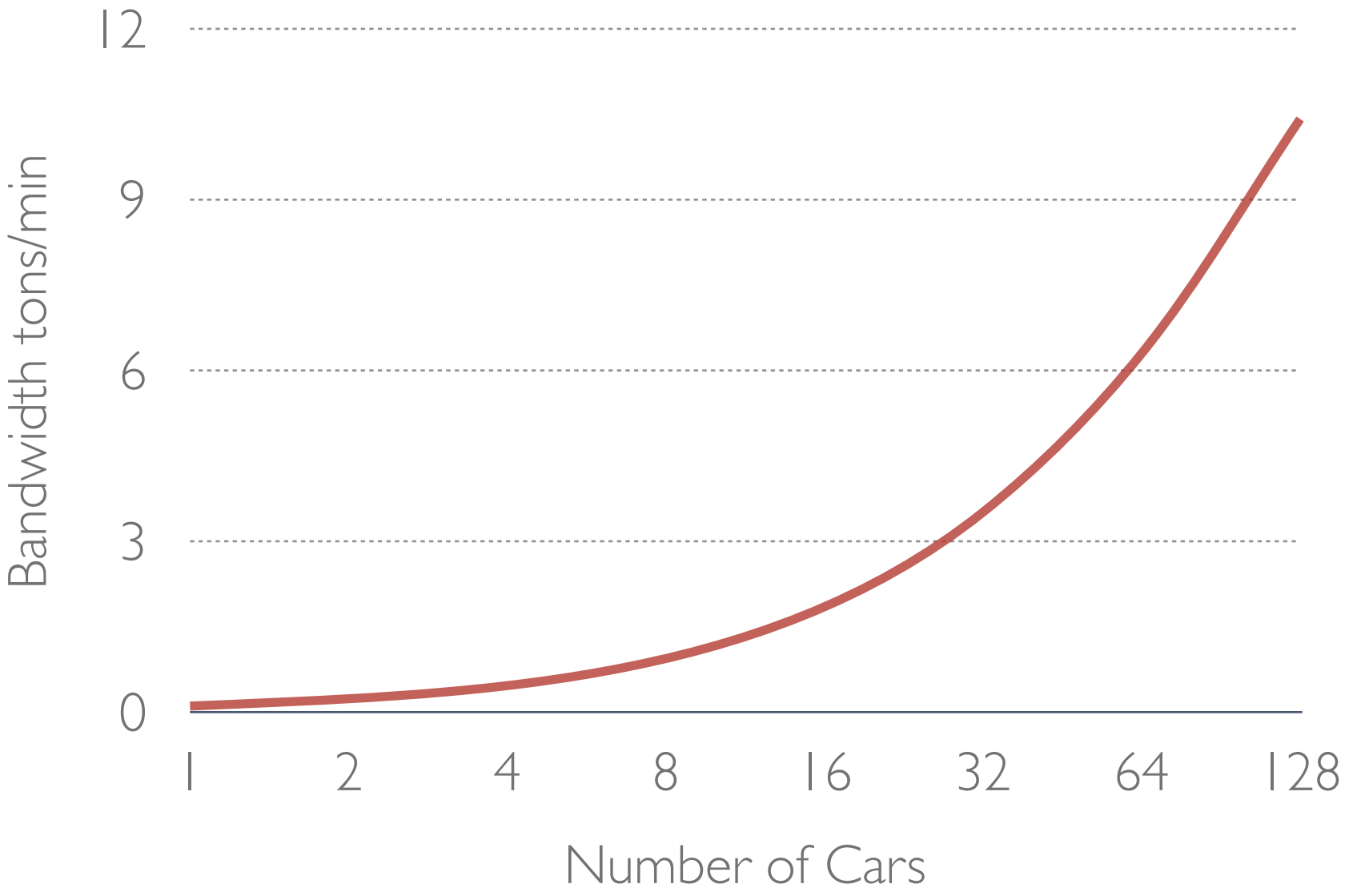
CC Flickr/Ben Salter

CC Flickr/ajmexico

TRAIN LATENCY (8 HOUR TRIP, 60 TON CARS, 60 SEC/CAR)

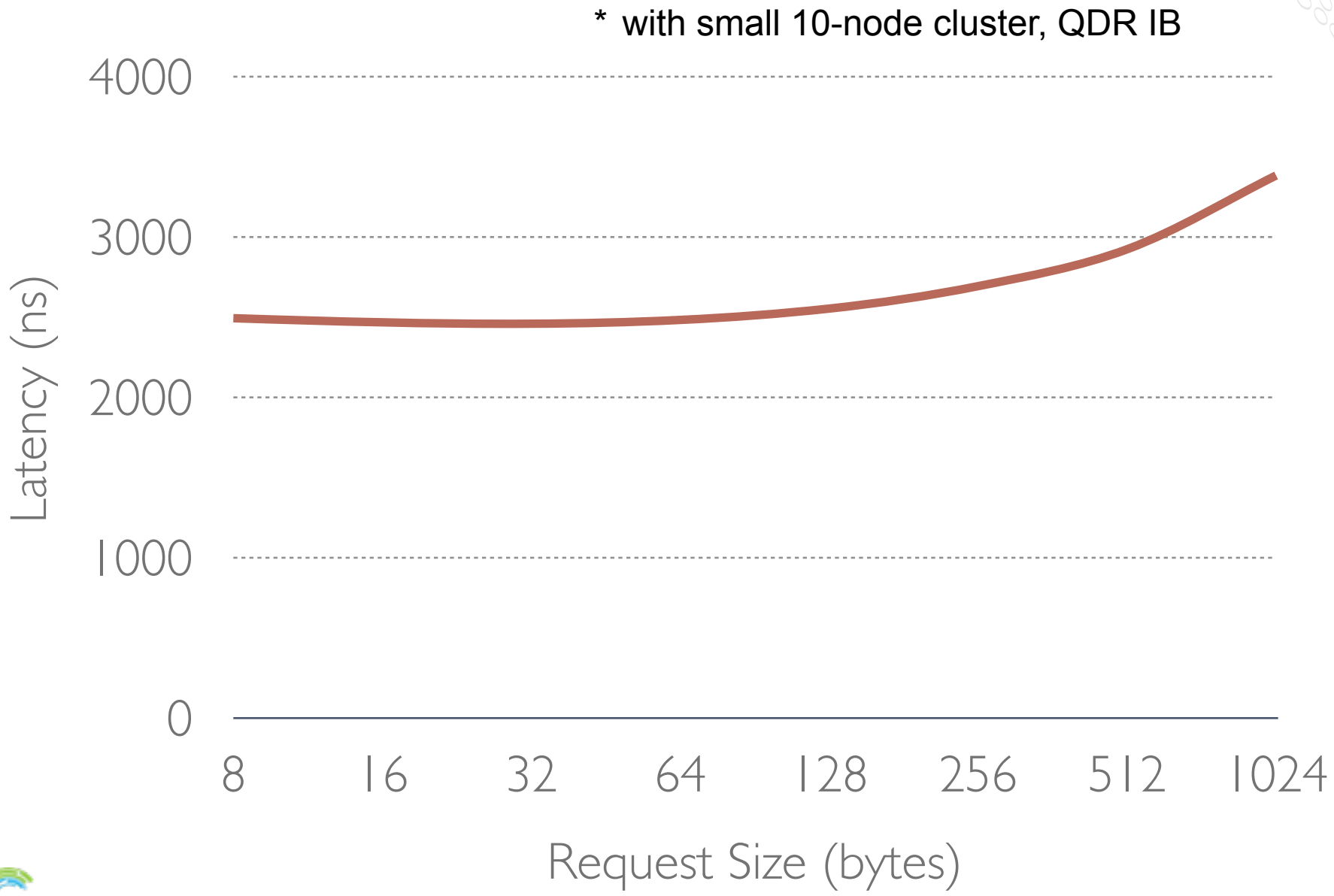


TRAIN BANDWIDTH

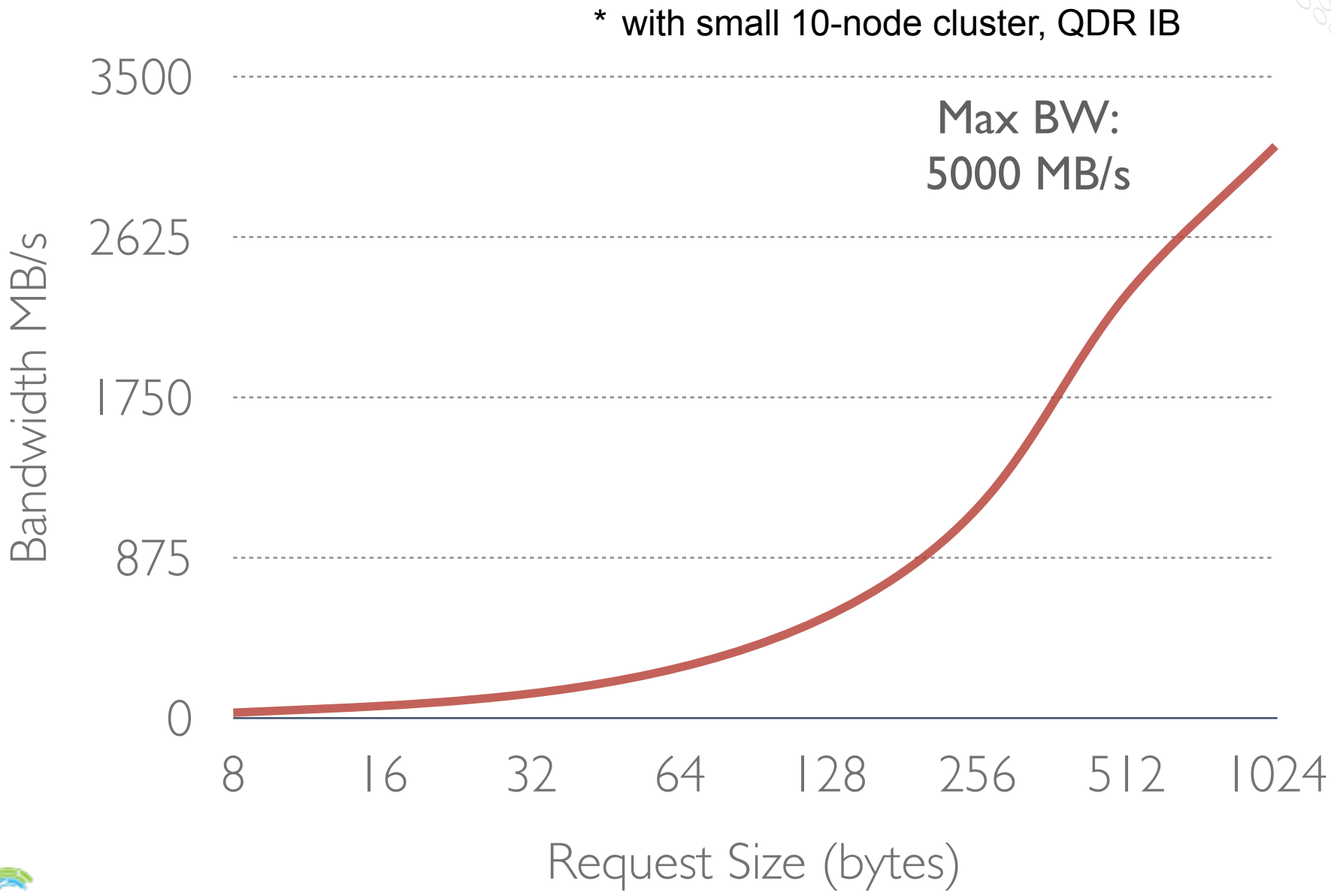


CC Flickr/ChrisDag

INFINIBAND (IB) LATENCY



INFINIBAND (IB) BANDWIDTH



AGGREGATION

CC Flickr/ajmexico

OVERLAP


CC Flickr/Barry Lewis

CACHE HELPS WITH BOTH!

BACKGROUND: MEMORY MODEL ALLOWS PREFETCH AND WRITE-BEHIND

Library of Congress

Memory model for `C11`, `C++11`, `Chapel`:

data race free programs are sequentially consistent

- See Adve, S.V., Boehm, H.-J. 2010. Memory models: a case for rethinking parallel languages and hardware. Communications of the ACM 53(8): 90–101. http://cacm.acm.org/magazines/2010/8/96610-memory-models-a-case-for-rethinking-parallel-languages-and-hardware/fulltext

A RACY PROGRAM

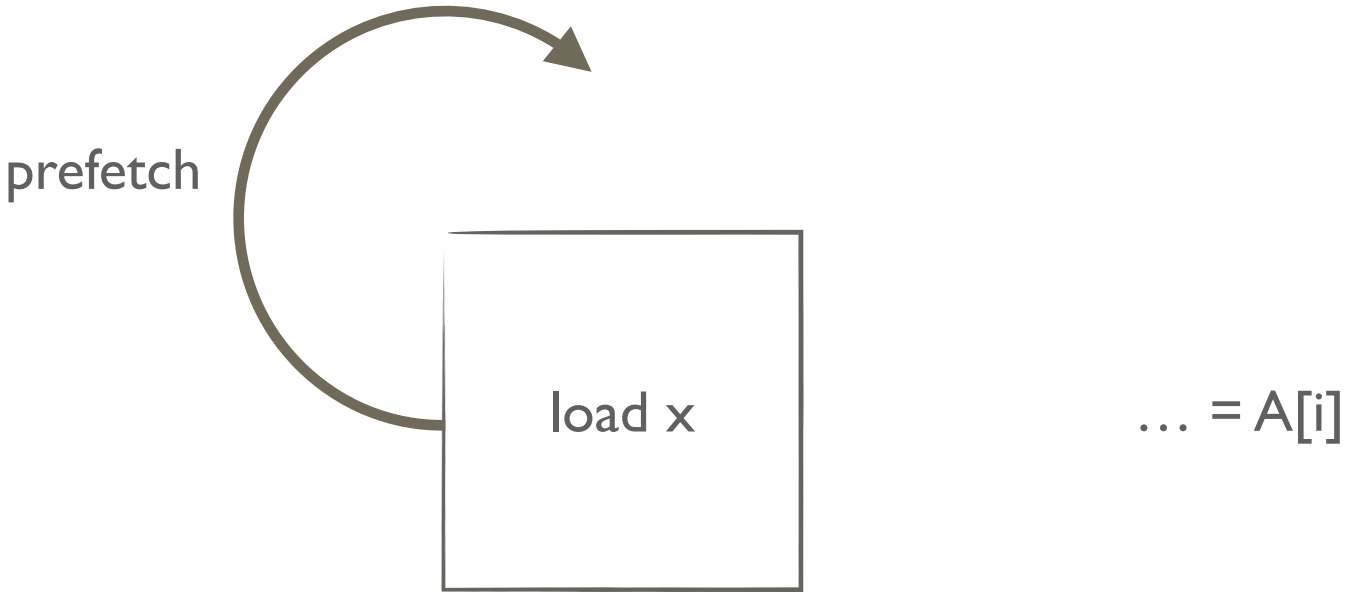
Thread 1	Thread 2
x = 42;	while 0 == notify { /* wait */ }
notify = 1;	compute_with(x);

A RACY PROGRAM

Thread 1	Thread 2
x = 42;	while 0 == notify { /* wait */ }
notify = 1;	compute_with(x);

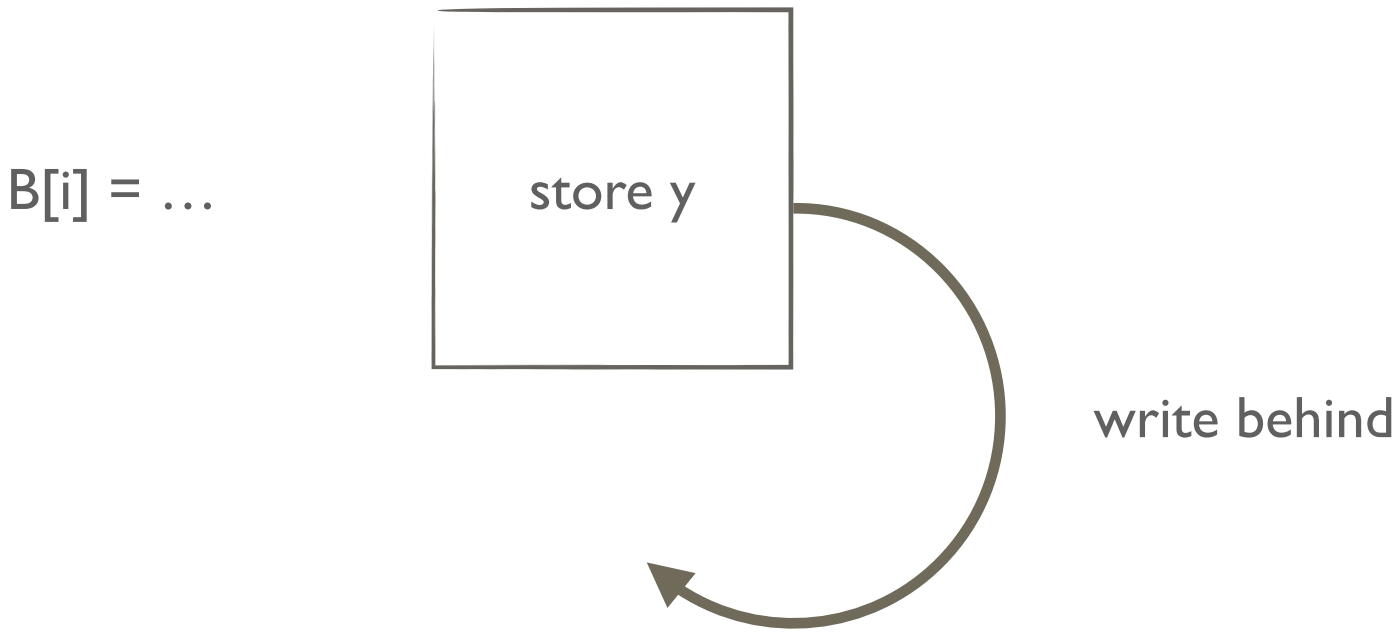
compiler *or* processor

Thread 1	Thread 2
r1 = 42;	r2 = notify; while 0 == r2 { /* wait */ }
notify = 1; x = r1;	compute_with(x);

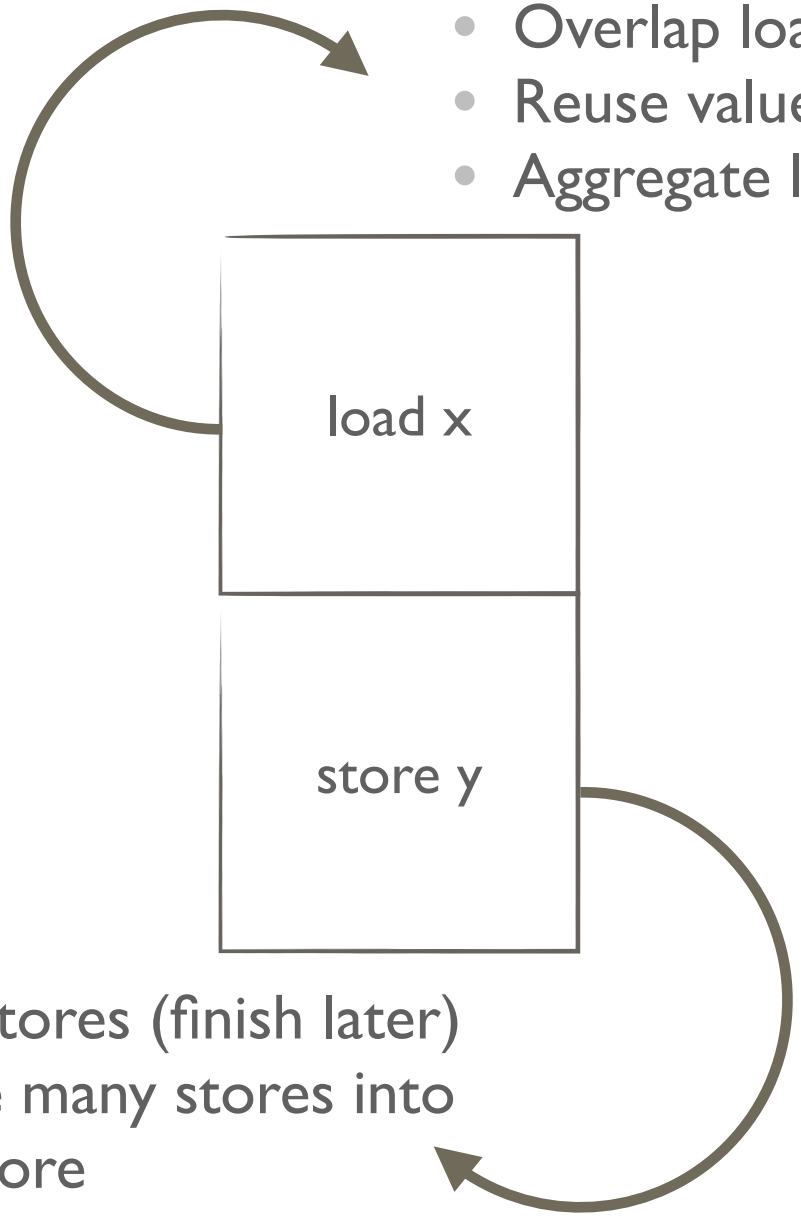


Compiler *and* processor would like to start loads earlier in order to hide memory latency. We'll call that *prefetch*.

Compiler *and* processor would like to complete stores later in order to hide memory latency. We'll call that *write behind*.



- Overlap loads (start early)
- Reuse values from earlier load
- Aggregate loads (cache lines)



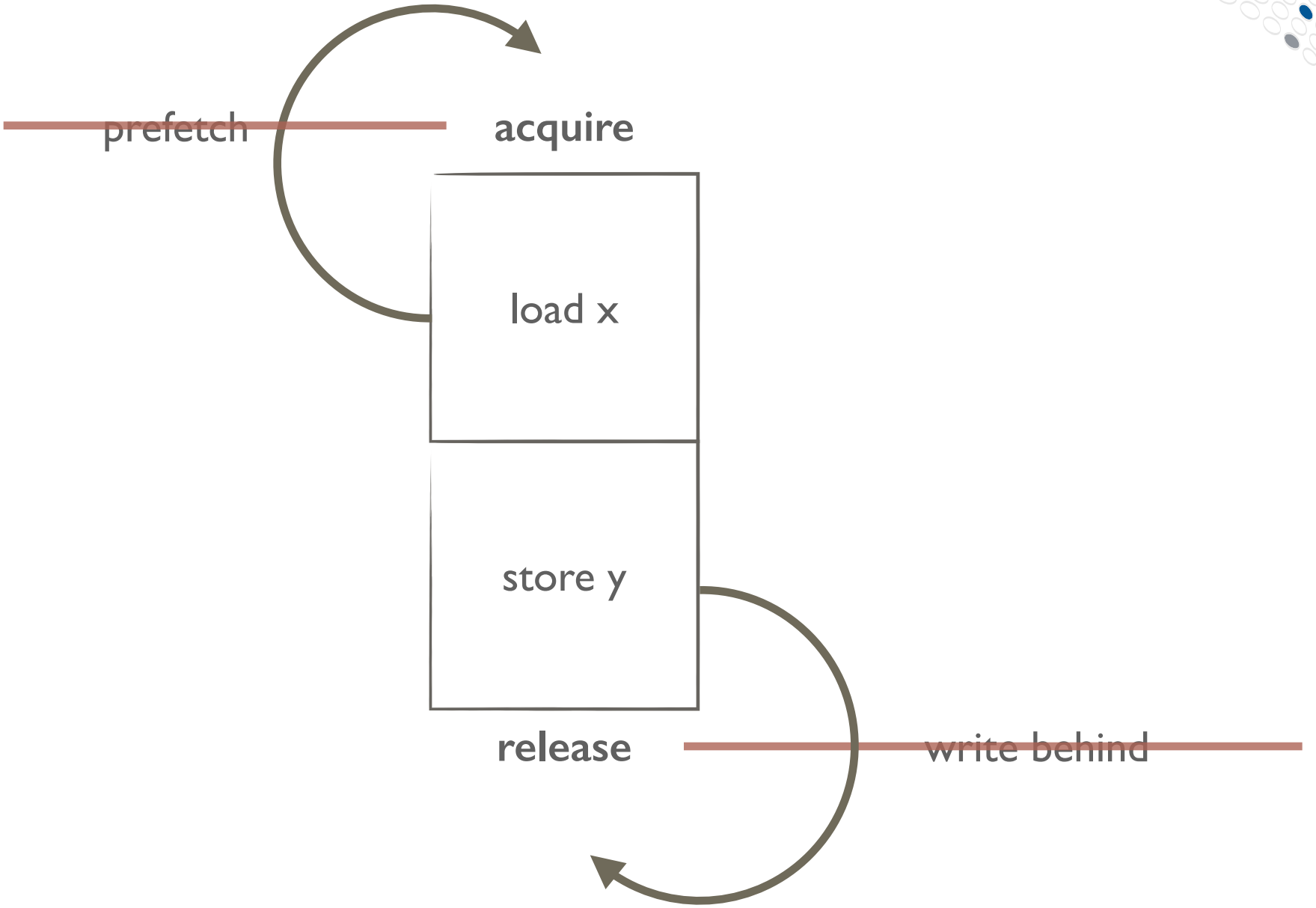
- Overlap stores (finish later)
- Aggregate many stores into a single store

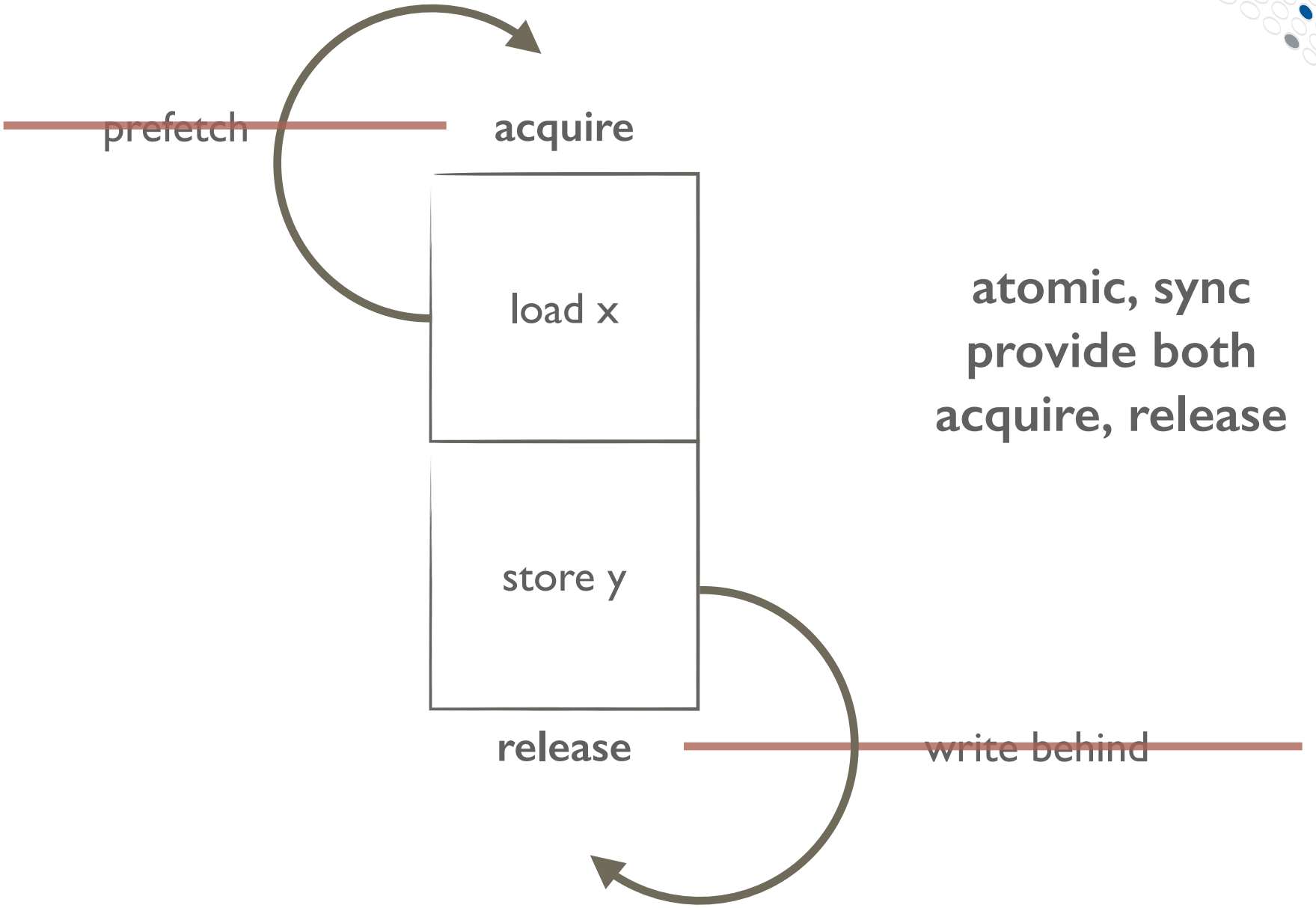
REMEMBER THE RACY PROGRAM?

Thread 1	Thread 2
x = 42;	while 0 == notify { /* wait */ }
notify = true;	compute_with(x);

compiler *or* processor

Thread 1	Thread 2
r1 = 42;	r2 = notify; while 0 == r2 { /* wait */ }
notify = 1; x = r1;	compute_with(x);


prefetch
acquire
load x
store y
release
write behind

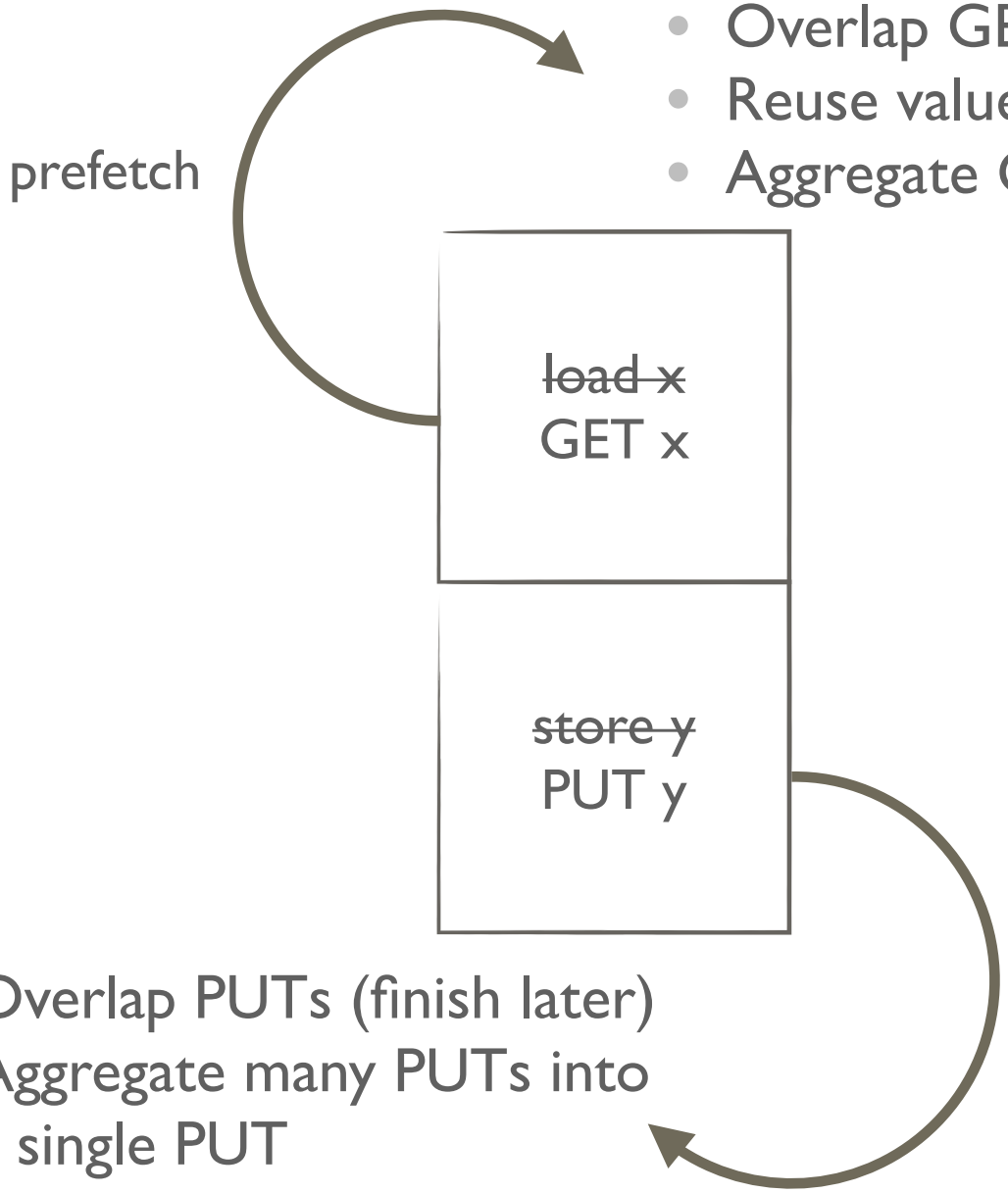


atomic, sync
provide both
acquire, release

COMMUNICATION OPTIMIZATION

Library of Congress

- Overlap GETs (start early)
- Reuse values from earlier GET
- Aggregate GETs (cache lines)



- Overlap PUTs (finish later)
- Aggregate many PUTs into a single PUT

FIXING IT WITH A CACHE

Library of Congress

CACHE FOR REMOTE DATA

- **Goal: communication aggregation and overlap**
- **Bonus points: avoiding repeated communication**

- **Software cache in Chapel's runtime**
- **One cache per pthread**
- **Write-back cache with dirty bits**

CACHE COHERENCY

- **Simple, local coherency**
- **Discard all cached data on *acquire***
- **Wait for pending operations on a *release***

- **Strategy used in related work with UPC**

CACHE FEATURES

	Overlap		Aggregation	
	GET	PUT	GET	PUT
Do PUTs in background		X		
Start one PUT per contiguous written region				X
Round GETs up to 64-byte cache lines			X	
Sequential read-ahead	X		X	
Programmer-provided prefetch hints*	X			

OTHER APPROACHES

Library of Congress

WEAK MEMORY CONSISTENCY?

```
1 x starts at 0;
  ...
  if someOption then
2   x = 2;
  if someOtherOption then
3   x = 3;
4 return x;
```

WEAK MEMORY CONSISTENCY?

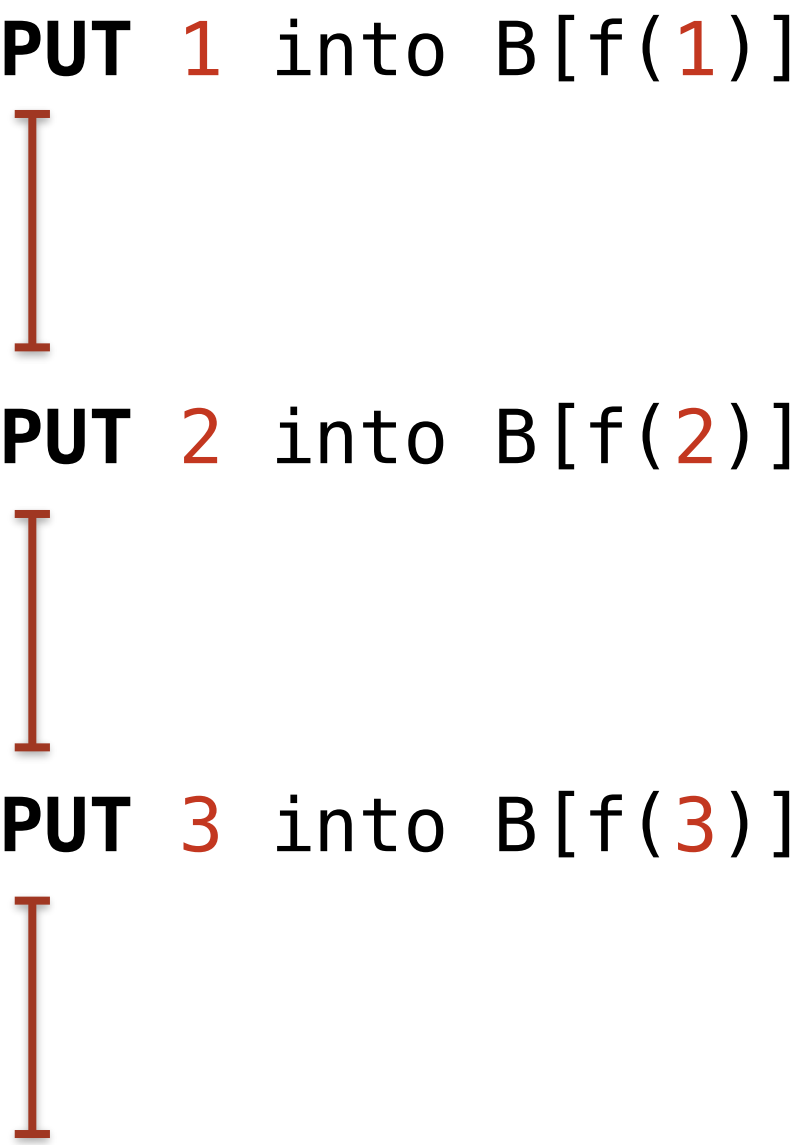
```
1 x starts at 0;
  ...
  ...
2   PUT 2 into x;
  ...
3   PUT 3 into x;
4 GET x;
```

Chapel	OpenSHMEM
result must be 3	**result could be 0, 2, or 3**

COMPILER OPTIMIZATION?

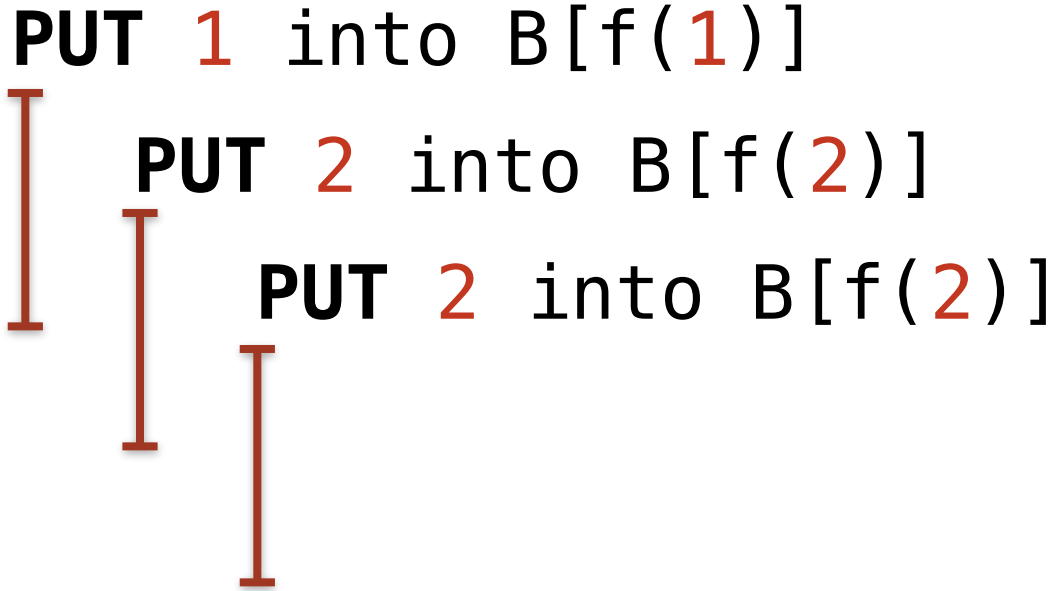
```
for i in 1..100
{
  // PUT into B
  B[f(i)] = i;
}
```

Can the compiler prove these PUTs do not overlap?



COMPILER OPTIMIZATION?

```
for i in 1..100
{
  // PUT into B
  B[f(i)] = i;
}
```



With a cache, conflicting access is handled at runtime.

OVERLAPPING GETS

```
var A:[1..n] int;
on Locales[1] {
  var sum:int;
  for i in 1..n do
    sum += A[f(i)]
}
```

We would like to overlap the GETs for A[f(i)] with each other

```
var A:[1..n] int;
on Locales[1] {
  var sum:int;
  var h: [0..k] handles;
  var bufs: [0..k] int;
  // Warm up loop
  for i in 1..k {
    // nonblocking GET A[f(i) into bufs[i%k]
    h[i%k] = get_nb(bufs[i%k], A[f(i)])
  }
  for i in 1..n {
    wait (h[i%k]);
    sum += bufs[i%k];
    if i+k<=n {
      // nonblocking GET A[f(i+k)] into bufs[(i+k)%k]
      h[(i+k)%k] = get_nb(bufs[(i+k)%k],A[f(i+k)])
    }
  }
}
```

Explicit overlap is messy!

```
var A:[1..n] int;
on Locales[1] {
  var sum:int;
  // Optional warm up
  for i in 1..k do prefetch(A[f(i)]);
  for i in 1..n {
    if i+k <= n then prefetch(A[f(i+k)]);
    sum += A[f(i)]
  }
}
```

Much better!

COMMUNICATION AGGREGATION

```
for i in 1..n do
  B[i] = compute(i);
```

```
var localB:[1..n] int;
for i in 1..n do
  localB[i] = compute(i);
B = localB;
```

```
for i in 1..n do
  consume(A[i]);
```

```
var localA:[1..n] int = A;
for i in 1..n do
  consume(localA[i]);
```

Simple, cache aggregates

Manual optimization reduces portability

PERFORMANCE

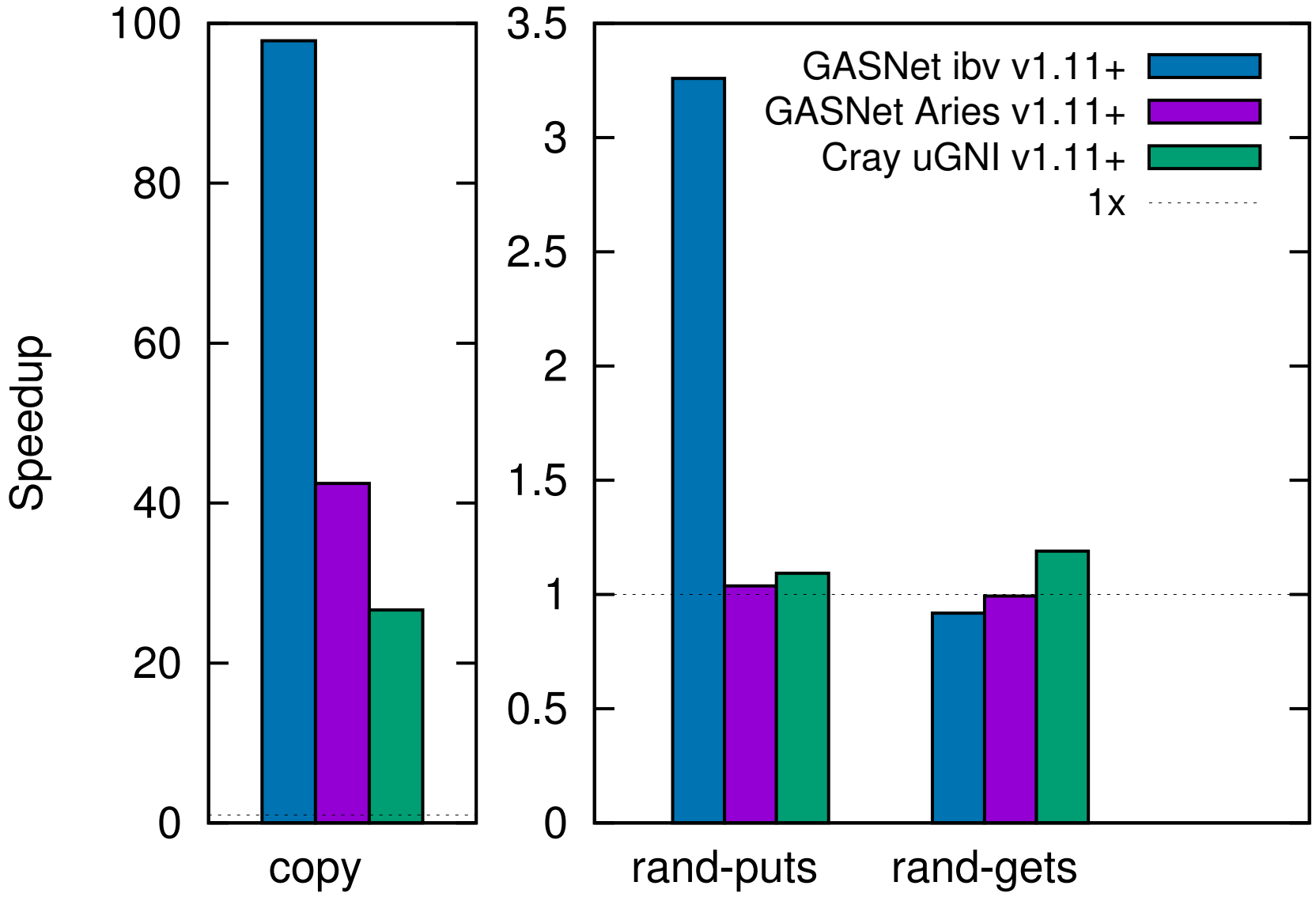
San Diego Air and Space Museum

TEST CONFIGURATIONS

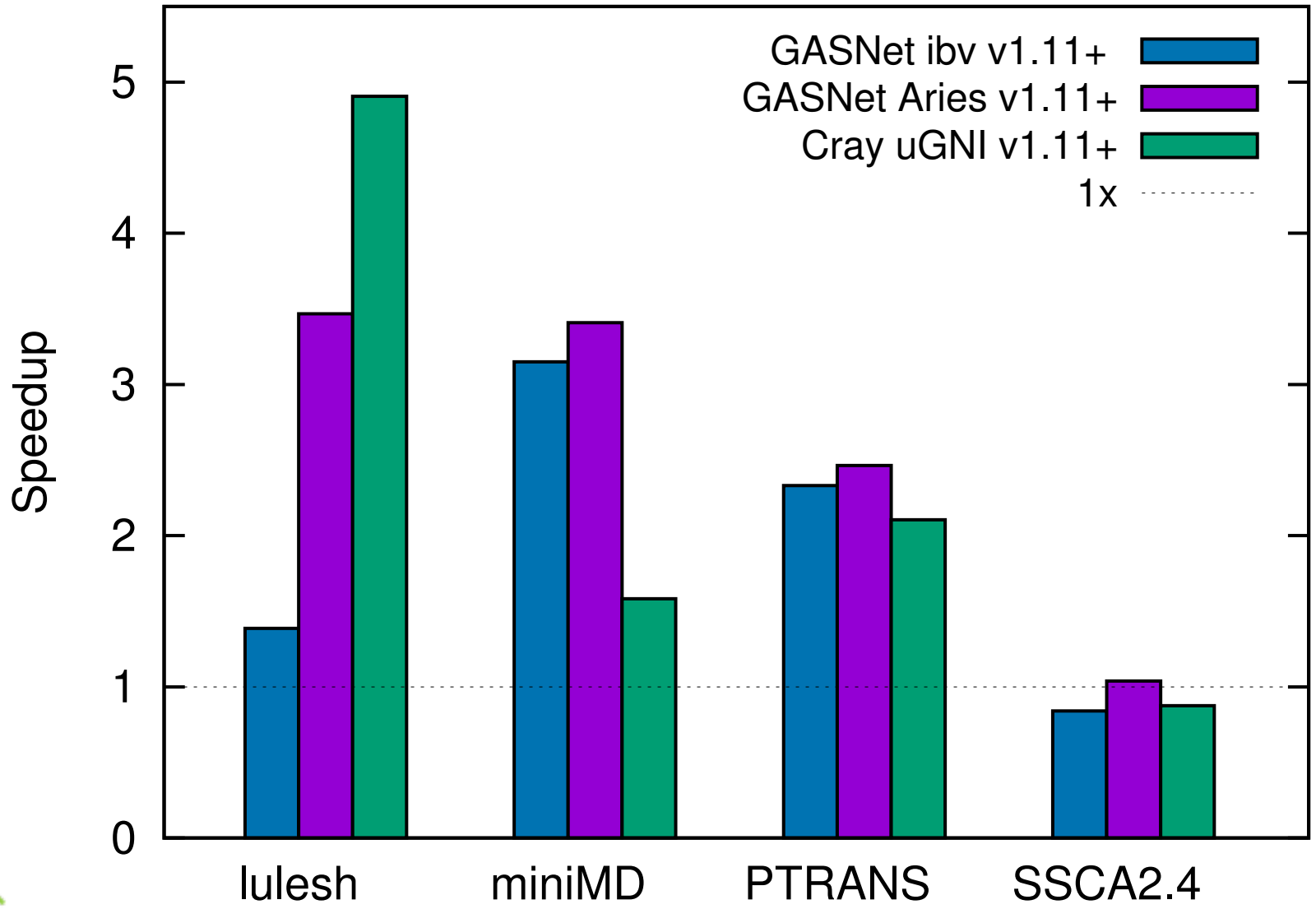
- Cray XC30™ system with 50 nodes, Aries network
  - GASNet Aries: GASNet with the aries conduit
  - Cray uGNI: native uGNI support for Chapel
- Cray CS400™ system with 200 nodes, FDR InfiniBand
  - GASNet ibv: GASNet with the InfinBand Verbs conduit

- v1.9+ is revision 5ba6639
- v1.11+ is revision 6c635a1

SYNTHETIC BENCHMARKS



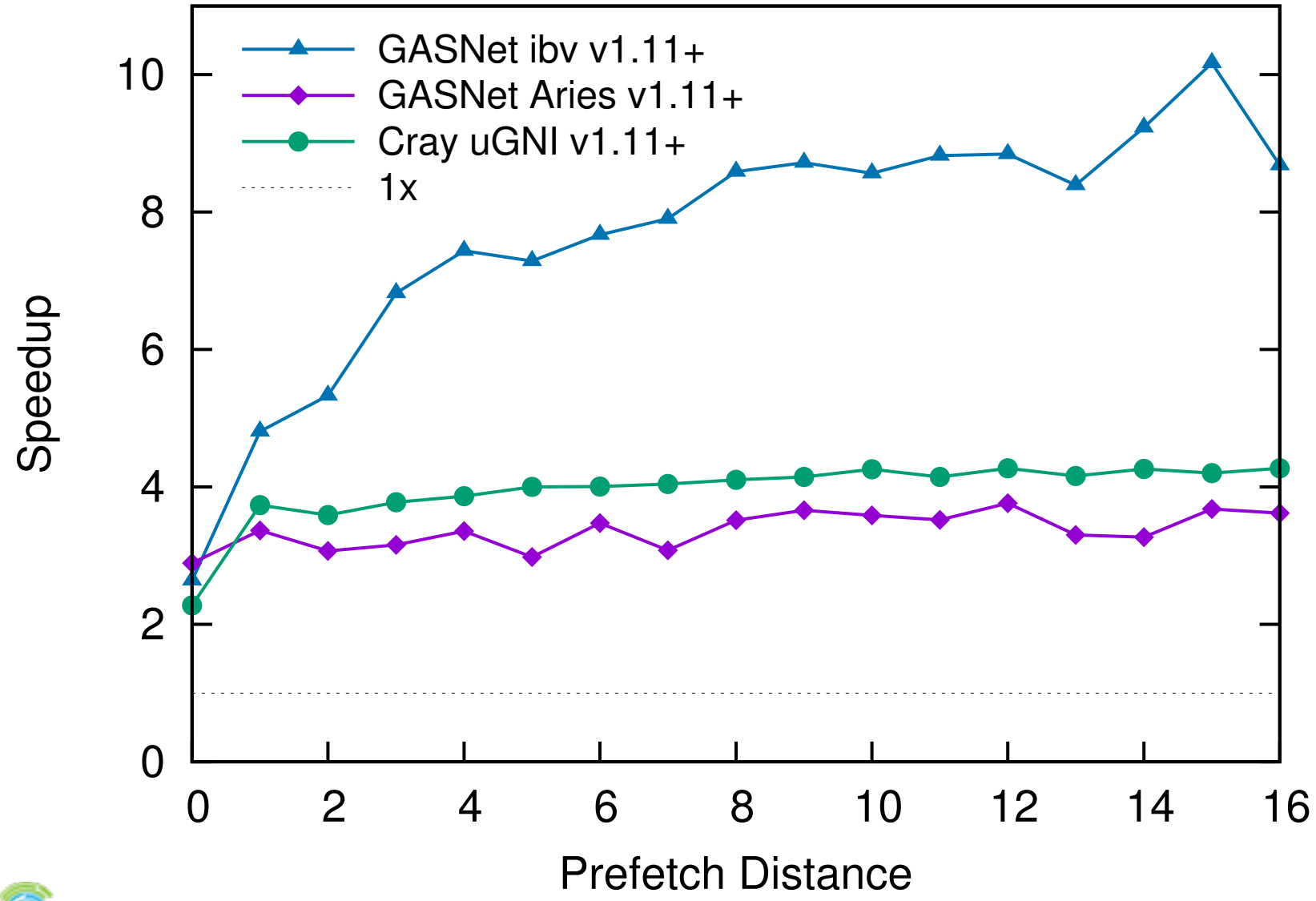
APPLICATION BENCHMARKS



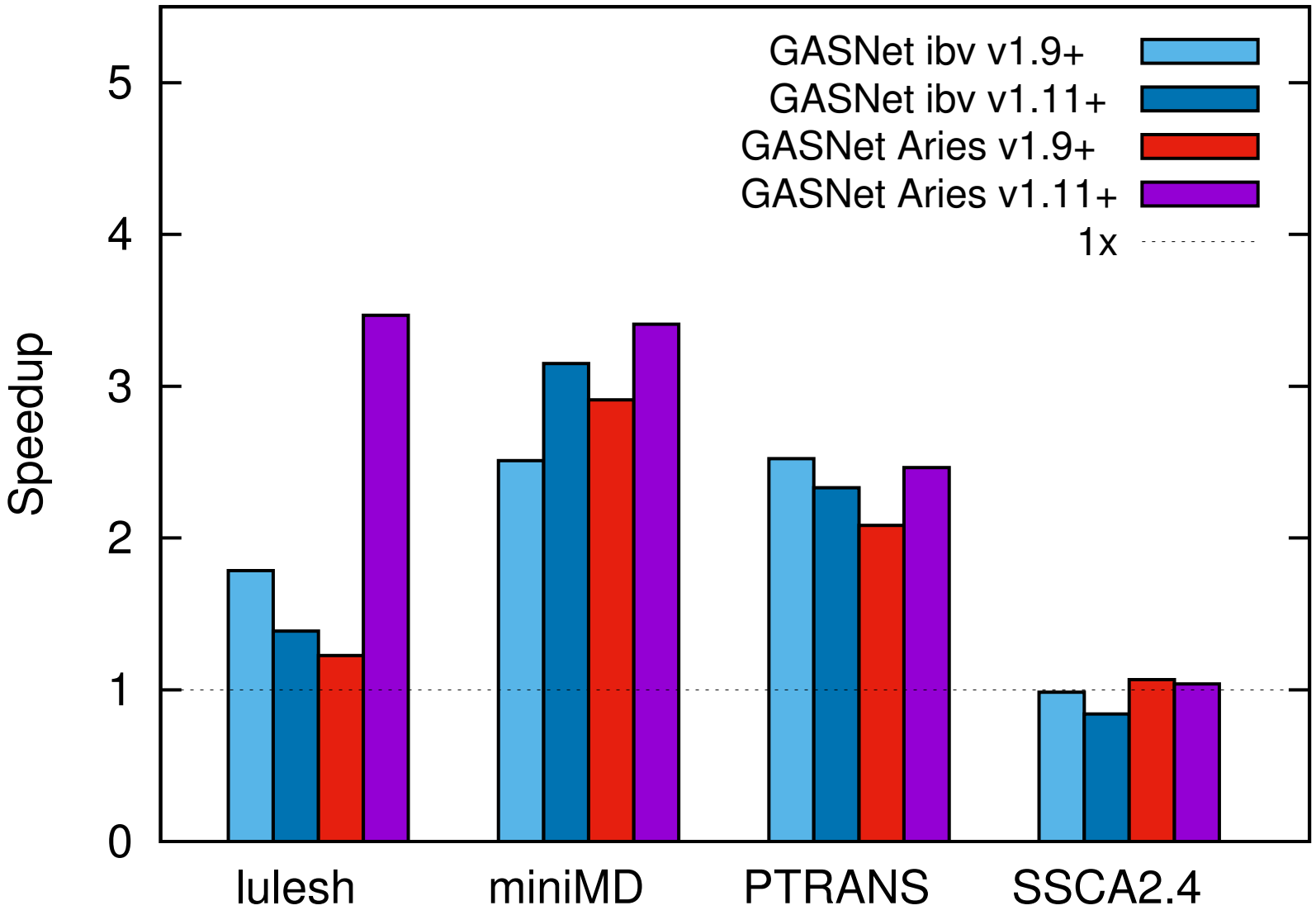
PREFETCH EXAMPLE

```
var A:[1..n] int;
on Locales[1] {
  var sum:int;
  // Optional warm up
  for i in 1..k do prefetch(A[f(i)]);
  for i in 1..n {
  if i+k <= n then prefetch(A[f(i+k)]);
      sum += A[f(i)]
  }
}
```

PREFETCH EXAMPLE

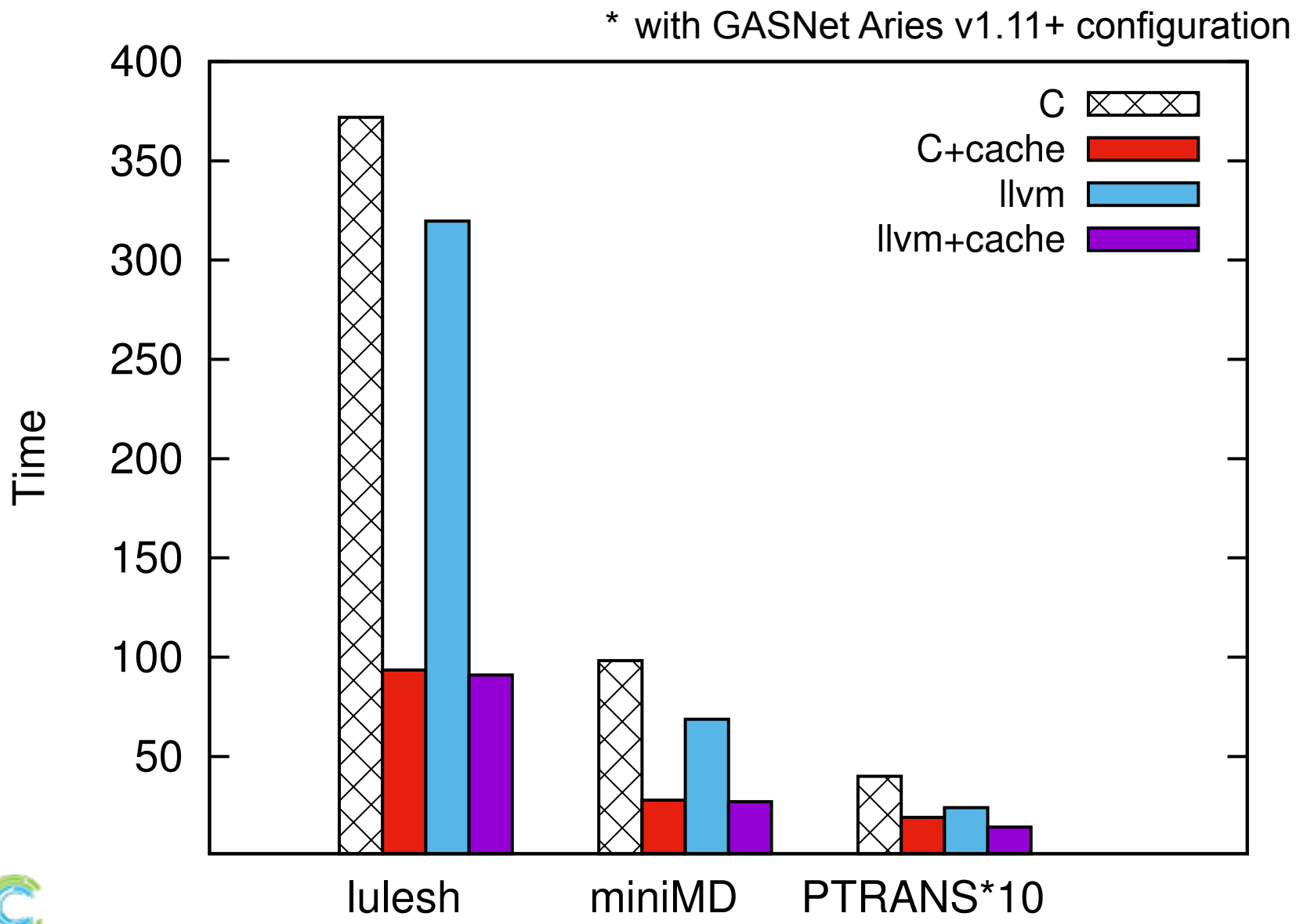


VS OPTIMIZATION



*for GASNet/Aries, lulesh improved 3.2x between v1.9+ and v1.11+

VS OPTIMIZATION



Cache for Remote Data: providing communication **overlap** and **aggregation** since Chapel v 1.10!

Library of Congress

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

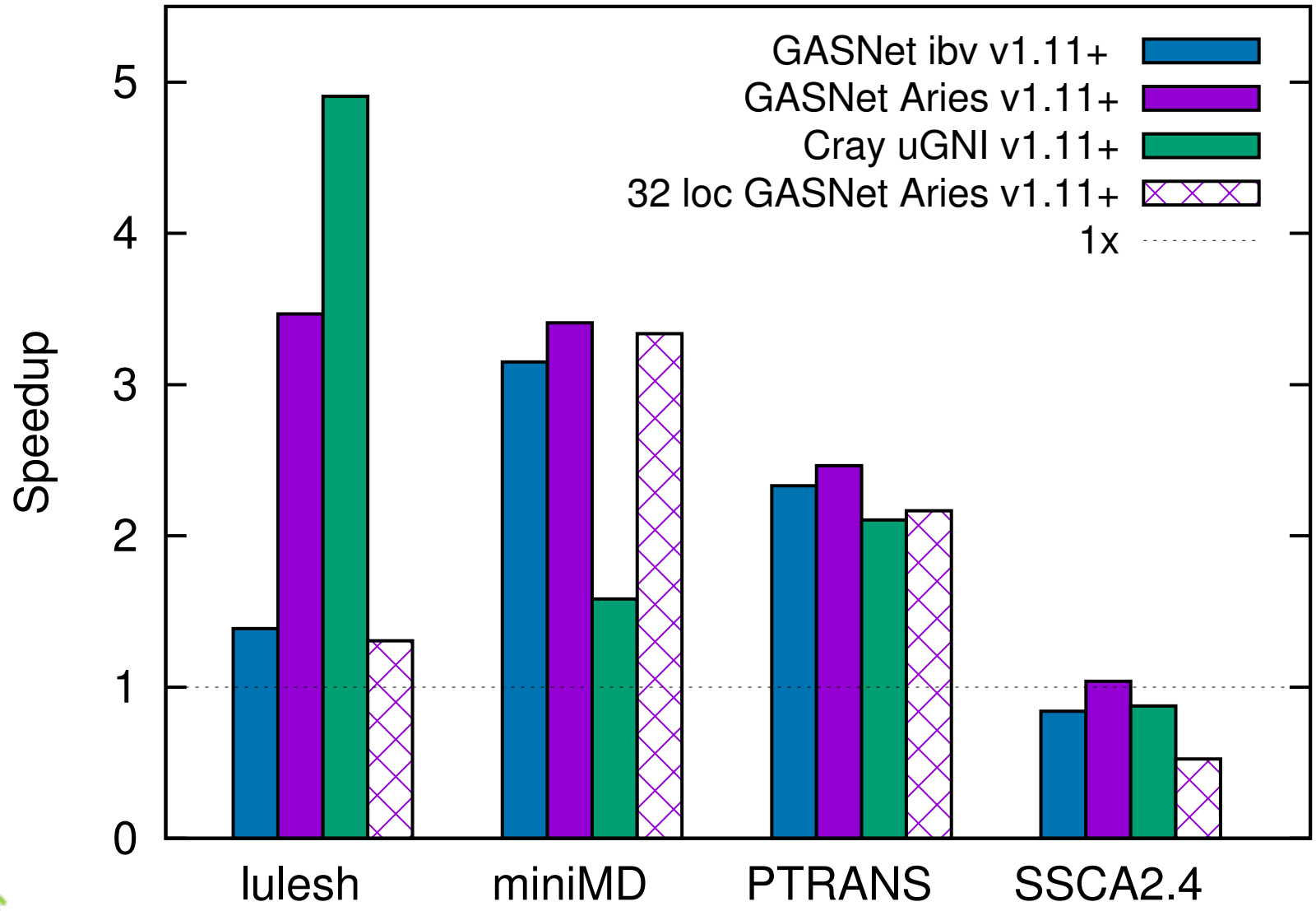
The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

CRAY
THE SUPERCOMPUTER COMPANY

http://chapel.cray.com chapel_info@cray.com https://github.com/chapel-lang/chapel/

Backup Slides

APPLICATION BENCHMARKS



ADDING IMPLIED FENCES

- ***acquire* and *release* triggered by task or on statement spawn, join, start, and finish**

```
release
on {
  acquire
  …
  release
}
acquire
```

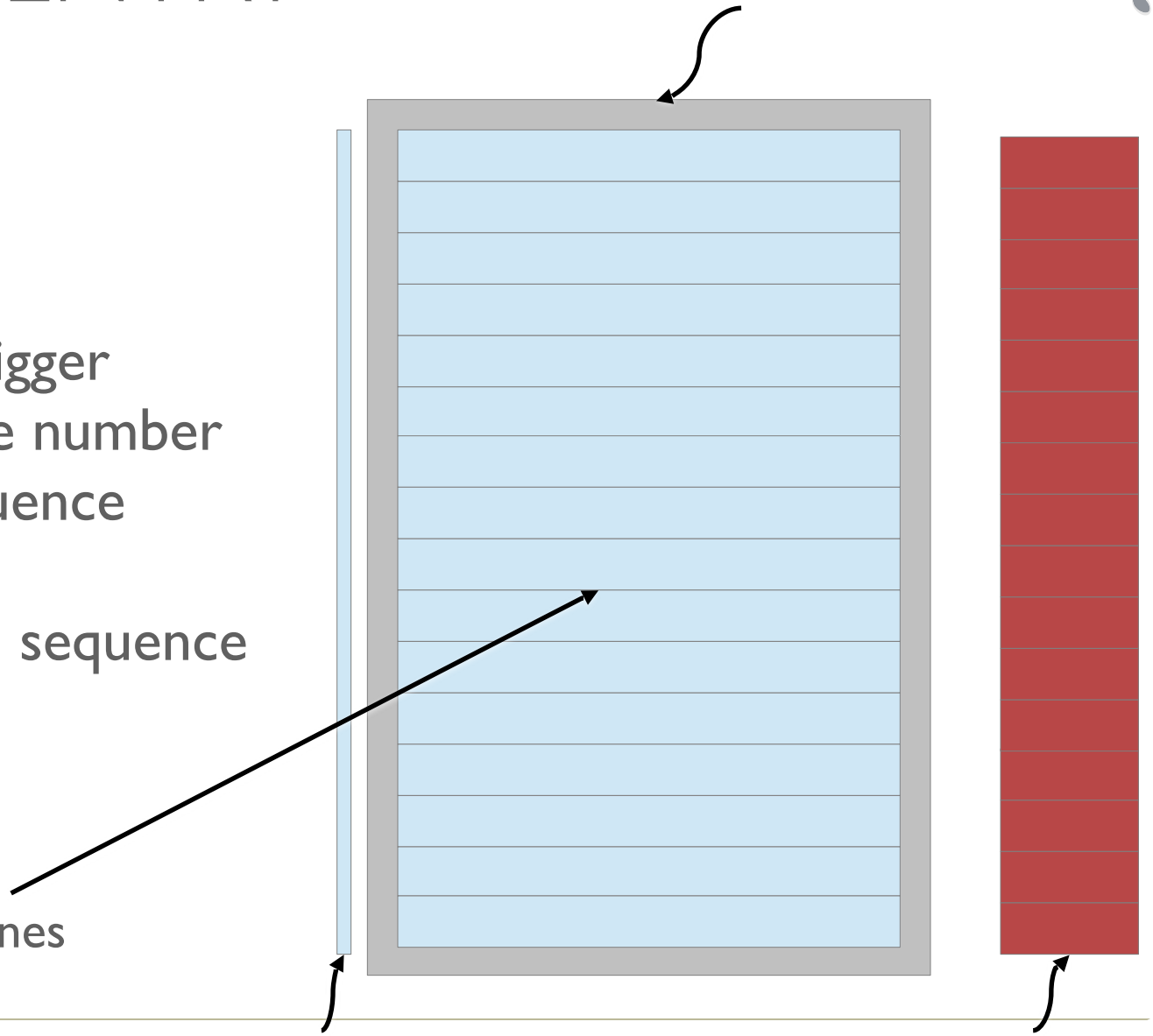
```
sync {
  release
  begin {
    acquire
    ….
    release
  }
} acquire
```

LOOKING INSIDE

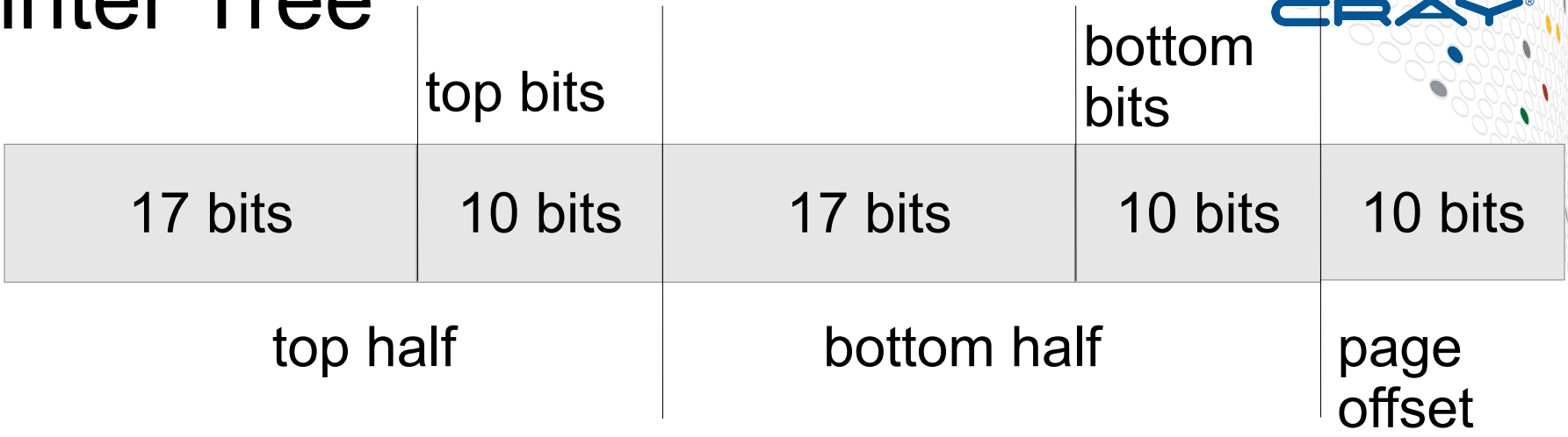
Library of Congress

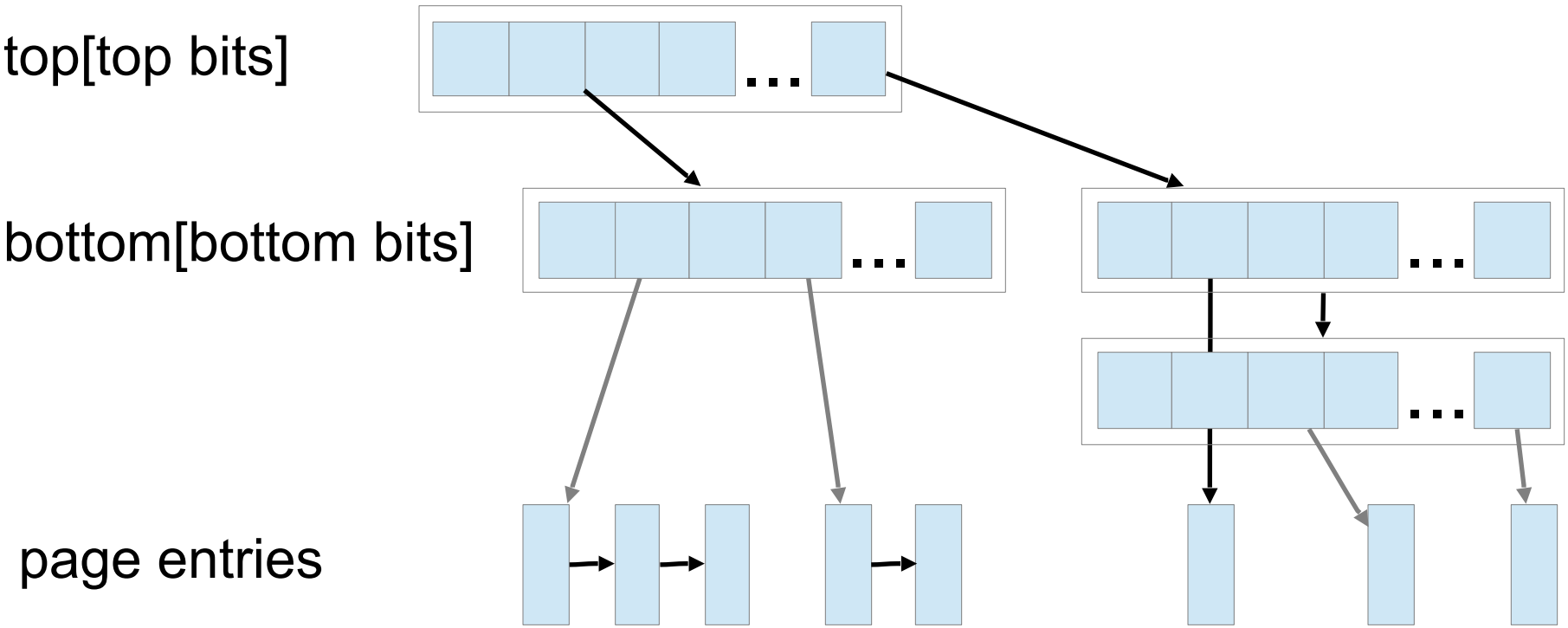
CACHE ENTRY

- node
- address
- readahead trigger
- min sequence number
- max put sequence number
- max prefetch sequence number



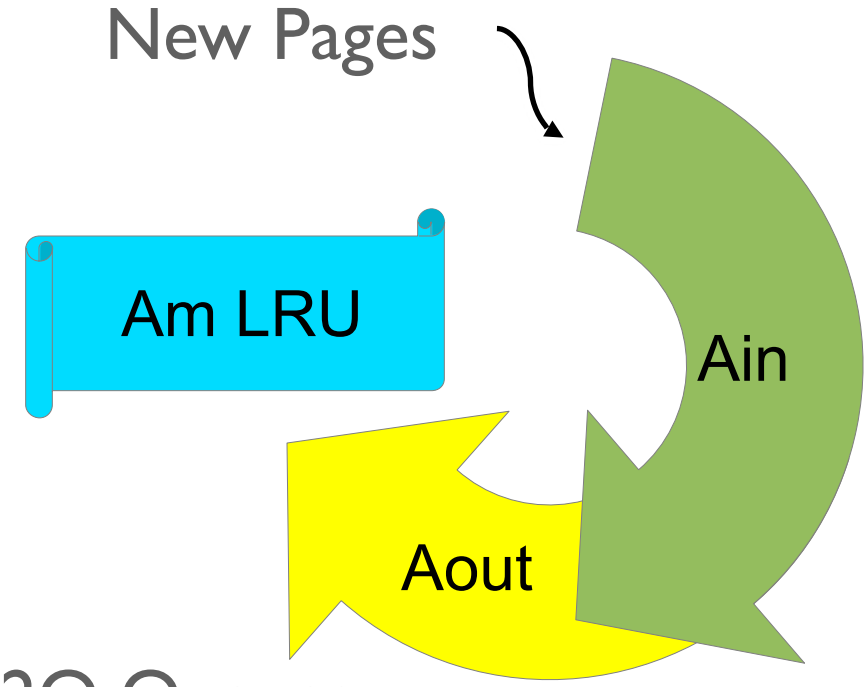
Pointer Tree



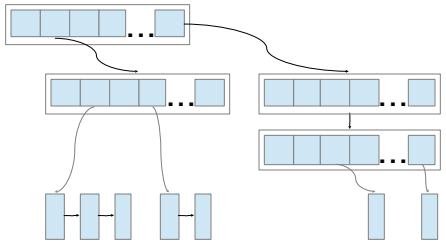


Inspired by “Two Level Tree Structure for Fast Pointer Lookup” by Hans J Boehm

CACHE DATA STRUCTURES



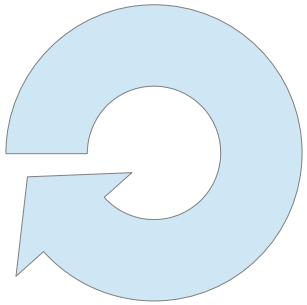
2Q Queues



Pointer Tree

Dirty LRU

Free Lists

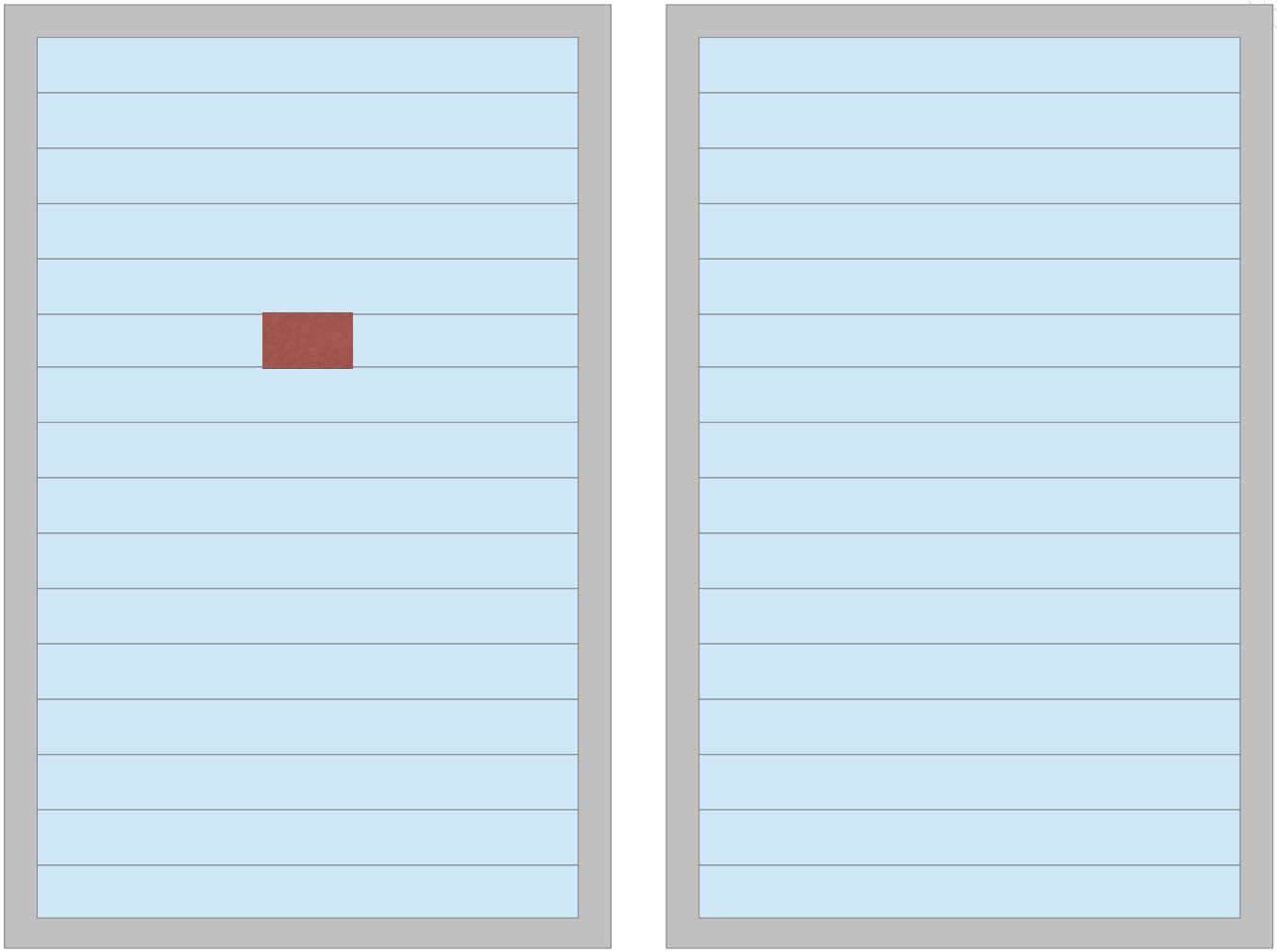


Operations Queue

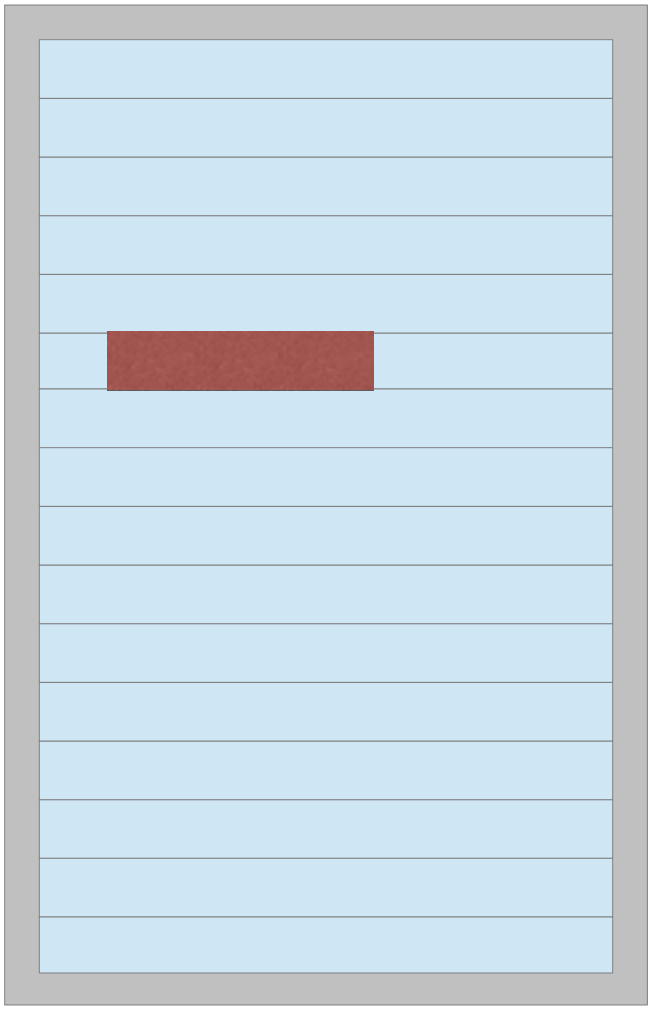
per task:

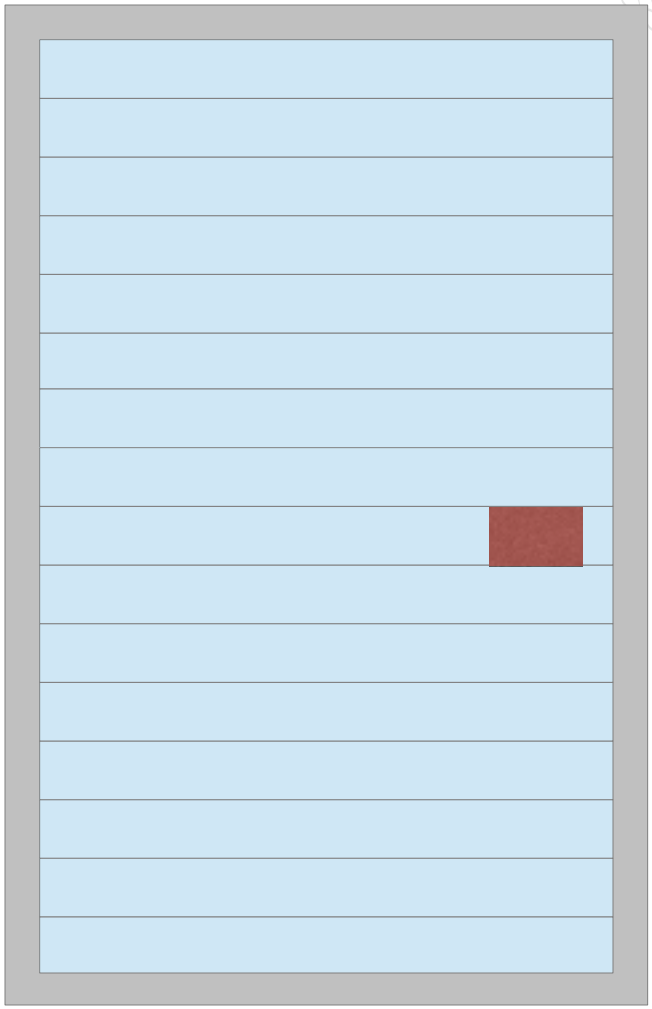
last acquire sequence number

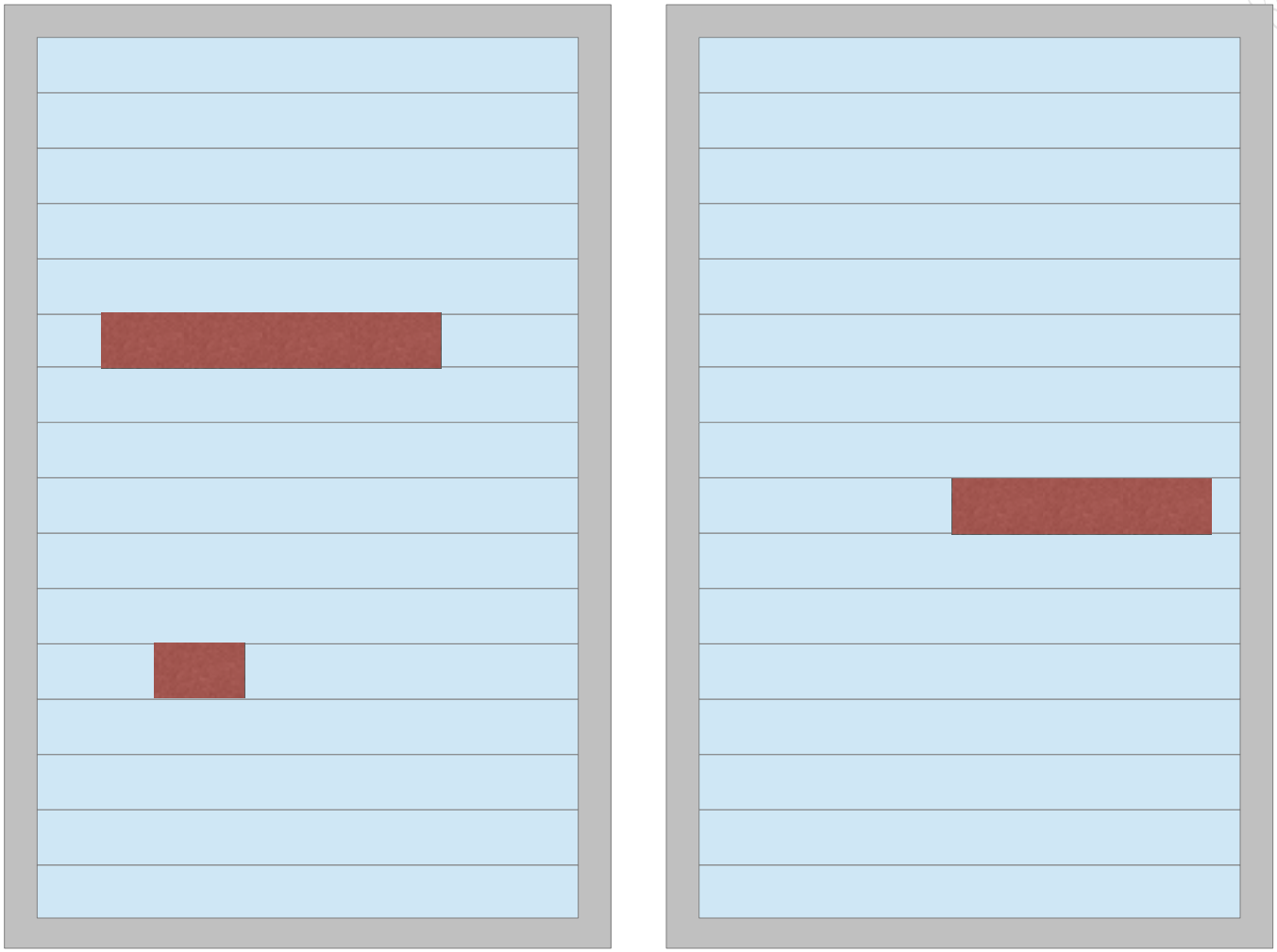
WRITE BEHIND



Write Recorded in Dirty Bits, Page added to Dirty Queue



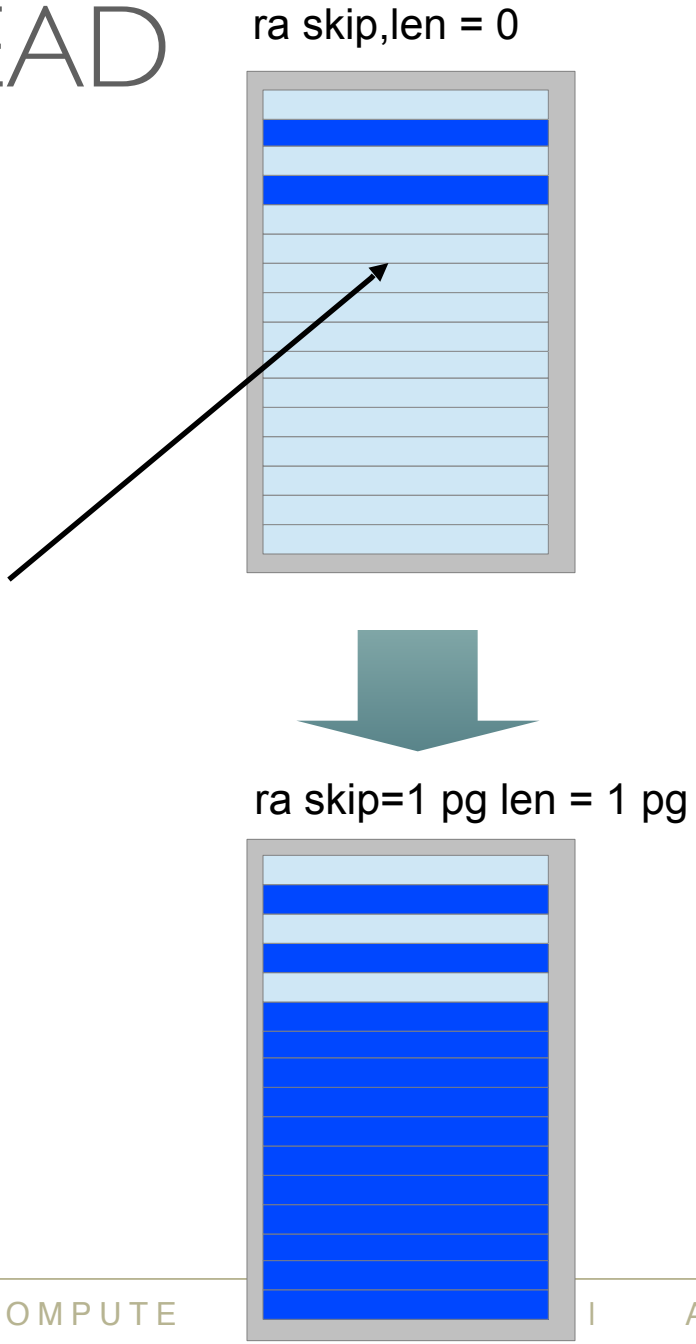


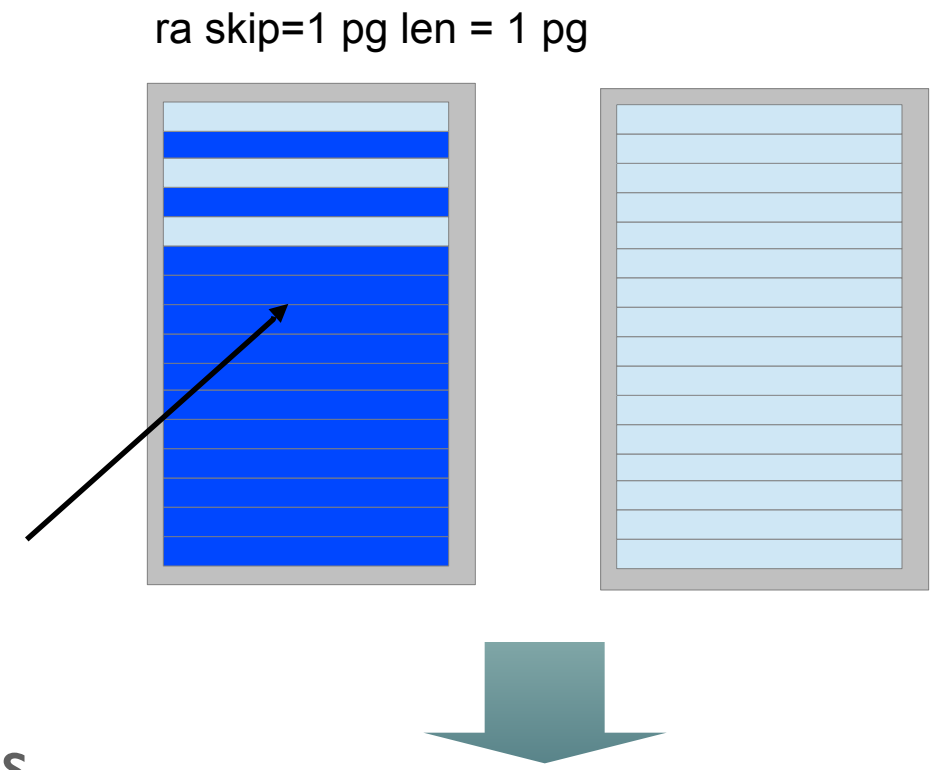


Flushed on *release* or
when there are too many dirty pages

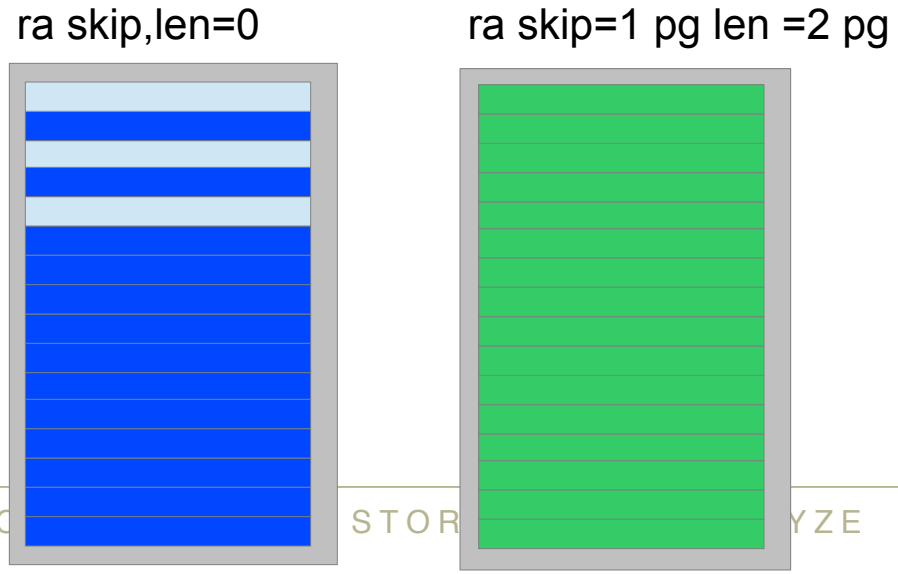
READAHEAD

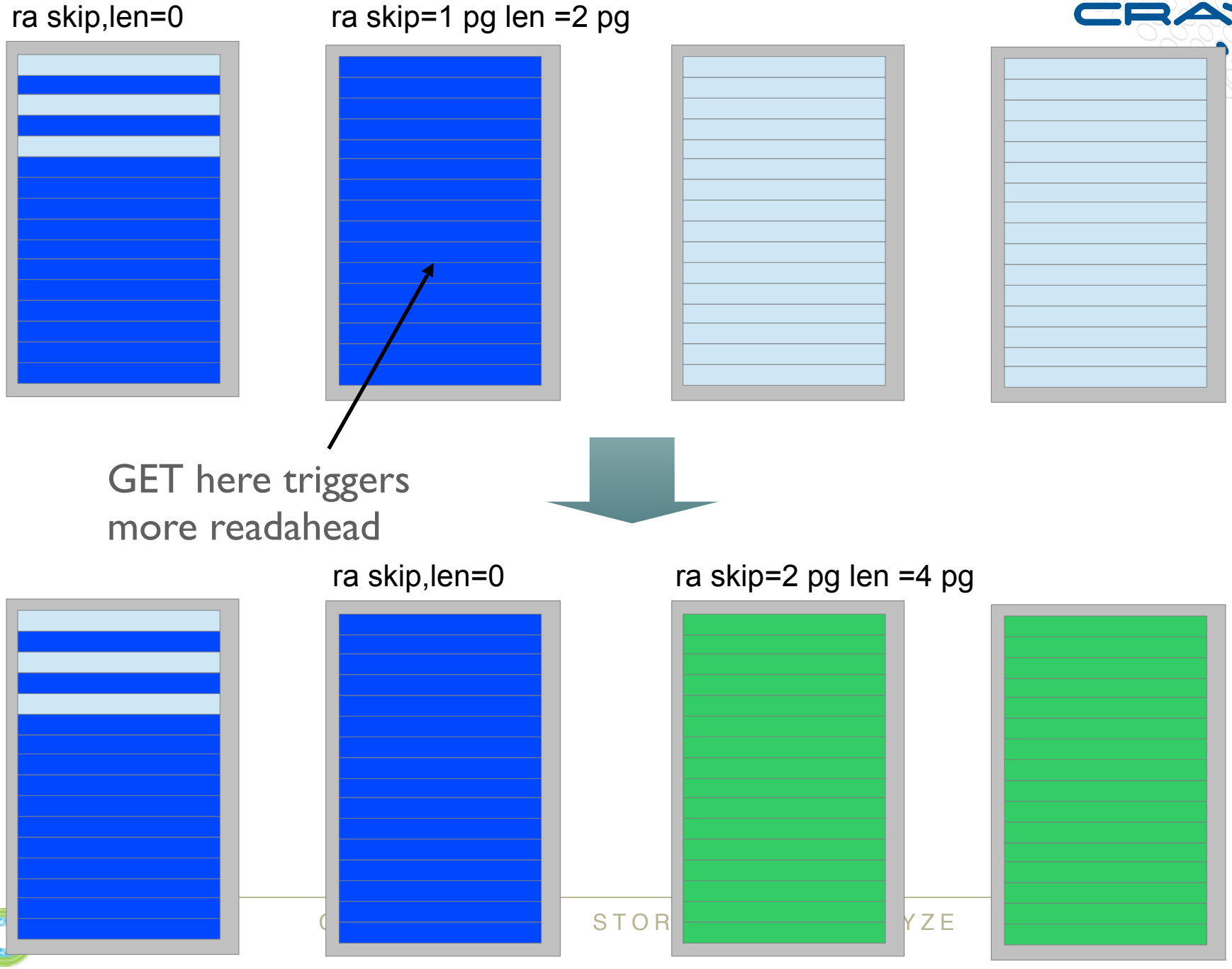
GET with 2 earlier valid lines triggers synchronous readahead



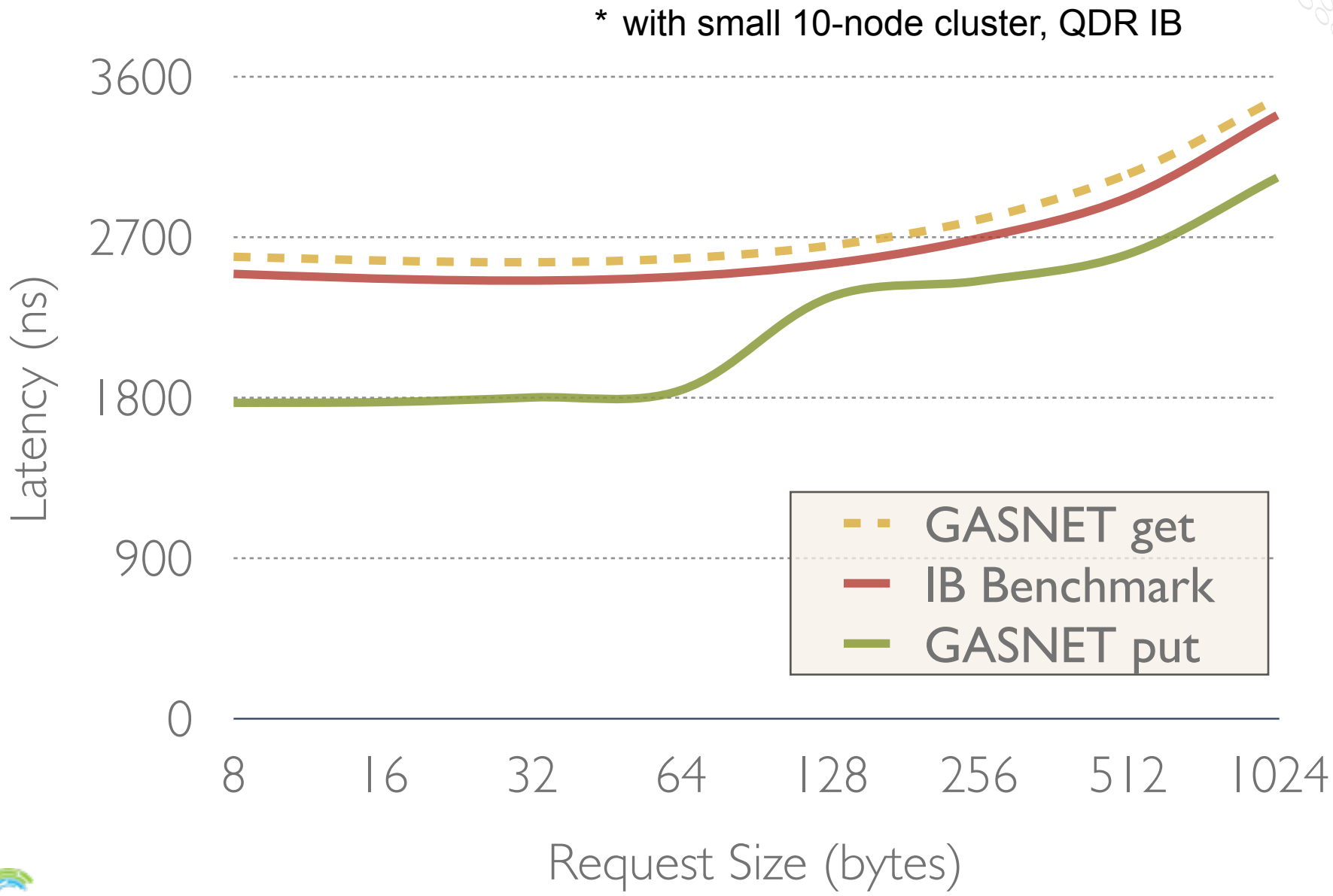


The next GET triggers asynchronous readahead

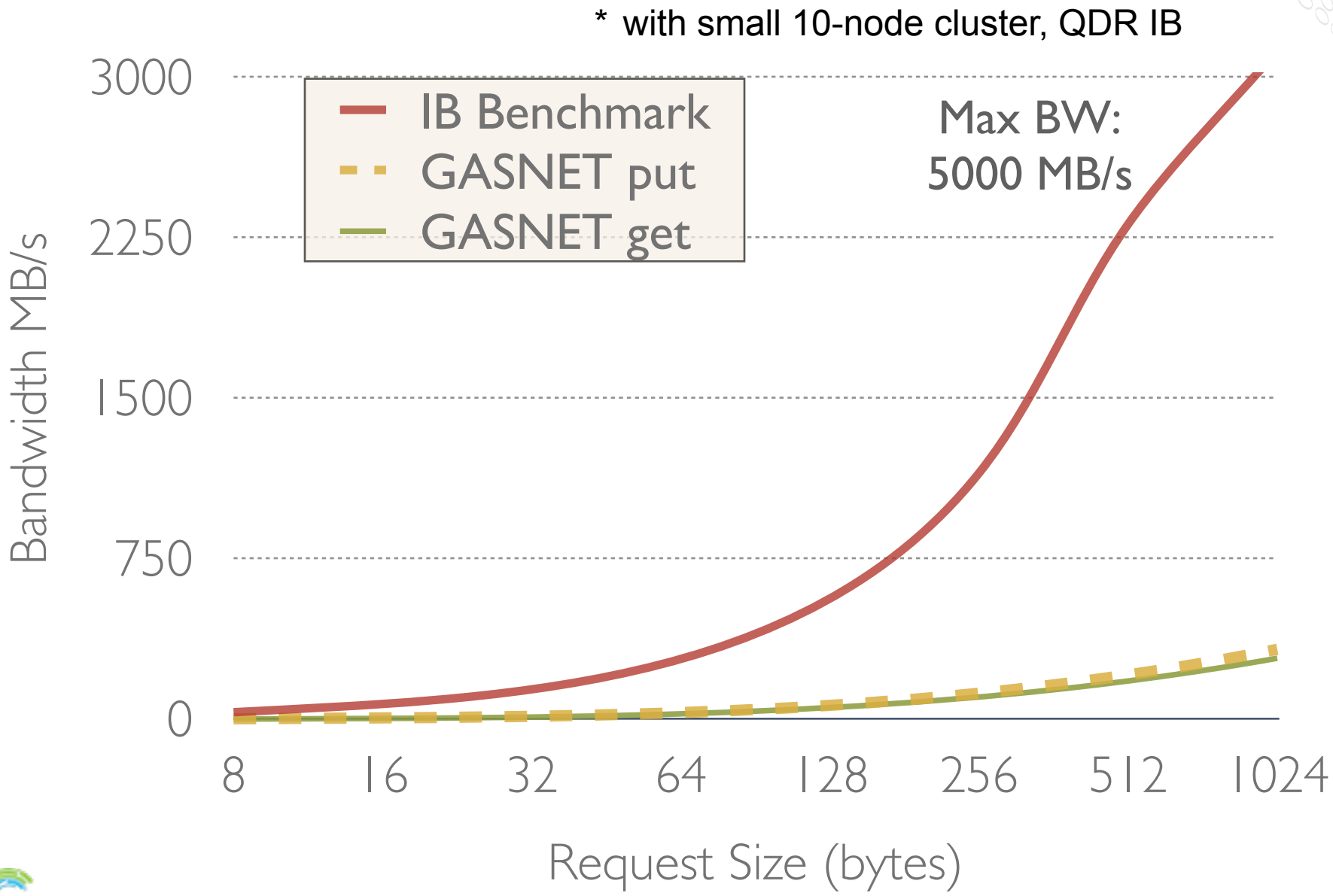



ra skip,len=0
ra skip=1 pg len =2 pg
GET here triggers
more readahead
ra skip,len=0
ra skip=2 pg len =4 pg

INFINIBAND (IB) LATENCY



INFINIBAND (IB) BANDWIDTH



Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

CRAY
THE SUPERCOMPUTER COMPANY

http://chapel.cray.com chapel_info@cray.com https://github.com/chapel-lang/chapel/