Chapel: HPCC Case Study

Steve Deitz

Cray Inc.

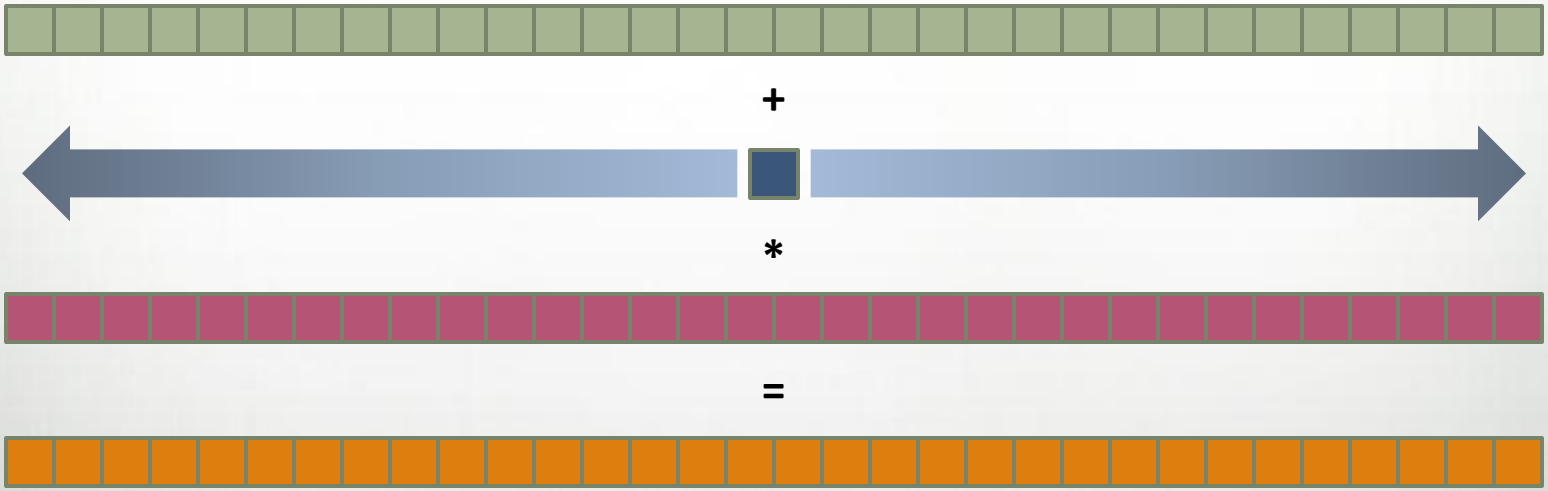
Outline

- HPCC STREAM Triad in Chapel
- HPCC RA in Chapel

Introduction to STREAM Triad

Given: *m*-element vectors *A*, *B*, *C*

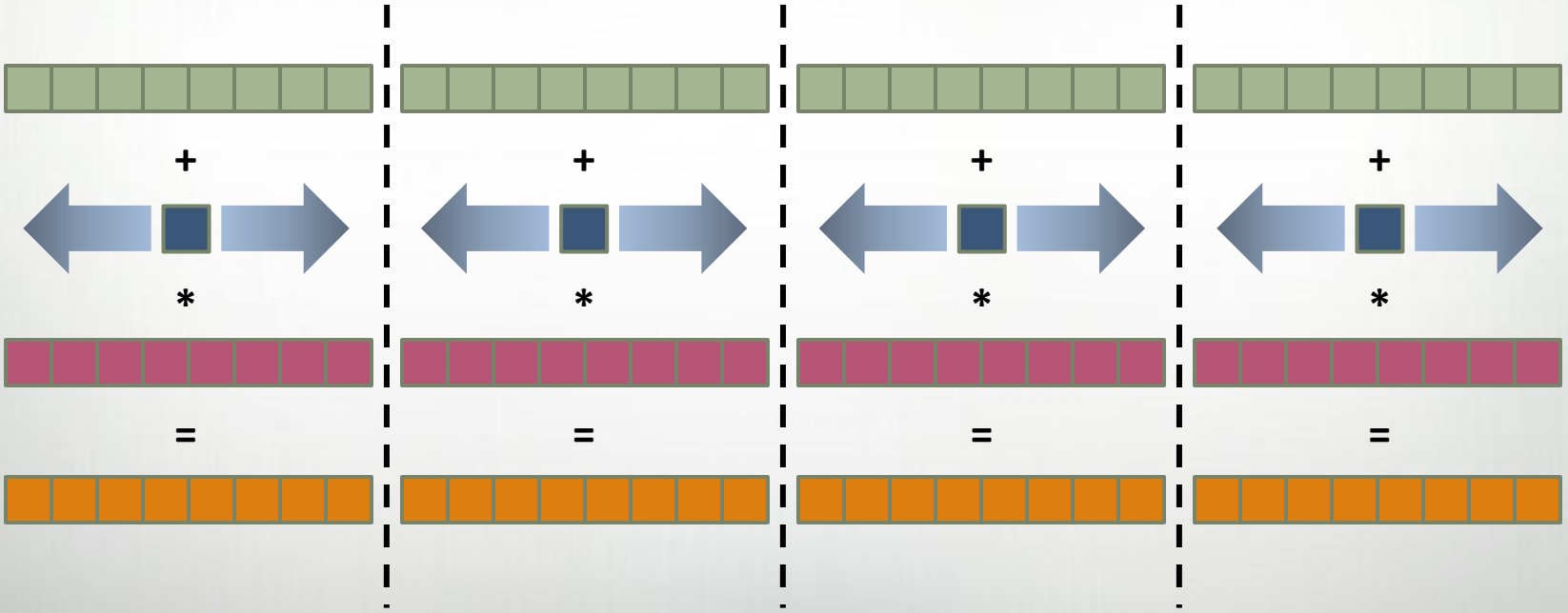
Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```



Distributed Parallelization of STREAM Triad

Given: *m*-element vectors *A*, *B*, *C*

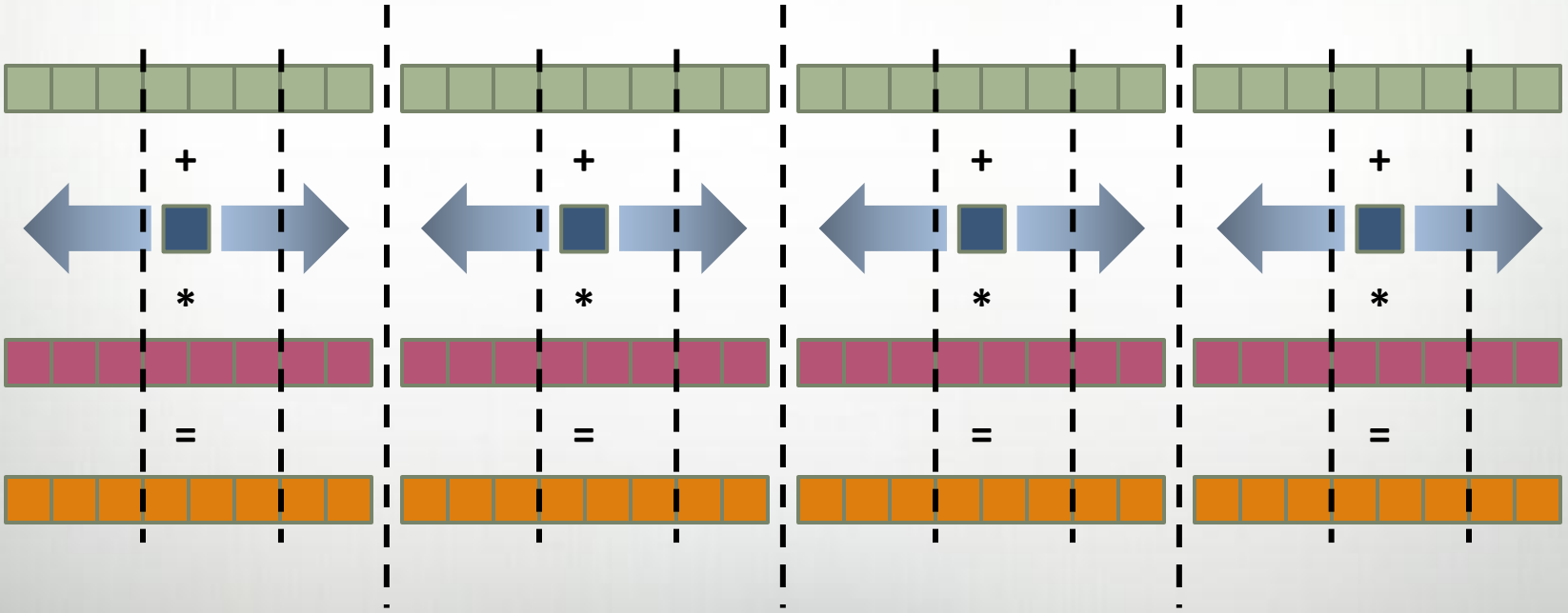
Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```



Further Parallelization of STREAM Triad

Given: *m*-element vectors *A*, *B*, *C*

Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```



STREAM Triad in Chapel: Single Locale

Given: *m*-element vectors *A*, *B*, *C*

Compute: `forall i in 1..m do A(i) = B(i) + α * C(i);`

```
config const m: int(64) = ...;
const alpha: real = 3.0;
const ProblemSpace: domain(1,int(64)) = [1..m];
var A, B, C: [ProblemSpace] real;

forall i in ProblemSpace do
  A(i) = B(i) + alpha * C(i);
```

STREAM Triad in Chapel: Single Locale

Given: *m*-element vectors *A*, *B*, *C*

Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```

```
config const m: int(64) = ...;
const alpha: real = 3.0;
const ProblemSpace: domain(1,int(64)) = [1..m];
var A, B, C: [ProblemSpace] real;

A = B + alpha * C;
```

More concise variation using whole array operations

STREAM Triad in Chapel: Single Locale

Given: *m*-element vectors *A*, *B*, *C*

Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```

```
config const m: int(64) = ...;
const alpha: real = 3.0;
const ProblemSpace: domain(1,int(64)) = [1..m];
var A, B, C: [ProblemSpace] real;

forall (a,b,c) in (A,B,C) do
  a = b + alpha * c;
```

Variation that iterates directly over the arrays

STREAM Triad in Chapel: Multi-Locale

Given: *m*-element vectors *A*, *B*, *C*

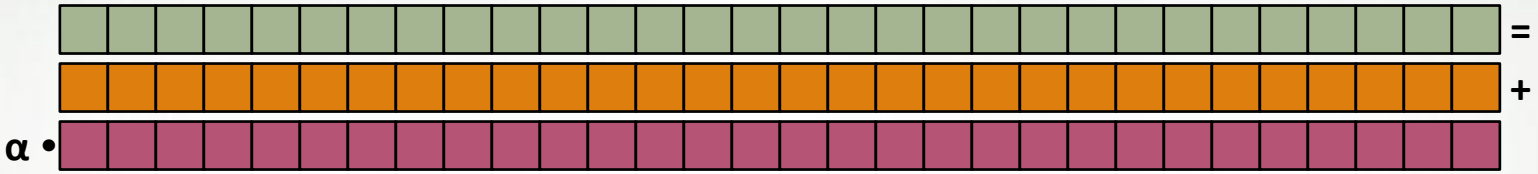
Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```

```
config const m: int(64) = ..., tpl = ...;
const alpha: real = 3.0;
const BlockDist = new Block(1,int(64),[1..m],tpl);
const ProblemSpace: domain(1, int(64))
                    distributed BlockDist = [1..m];
var A, B, C: [ProblemSpace] real;

forall (a,b,c) in (A,B,C) do
  a = b + alpha * c;
```

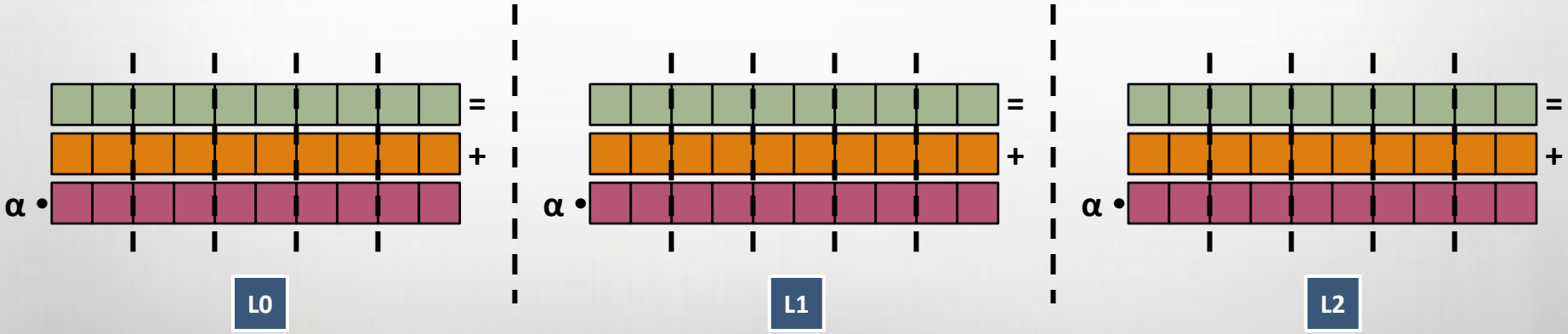
What is a Distribution?

A “recipe” for distributed arrays that...

Instructs the compiler how to Map the global view...



...to a fragmented, per-processor implementation



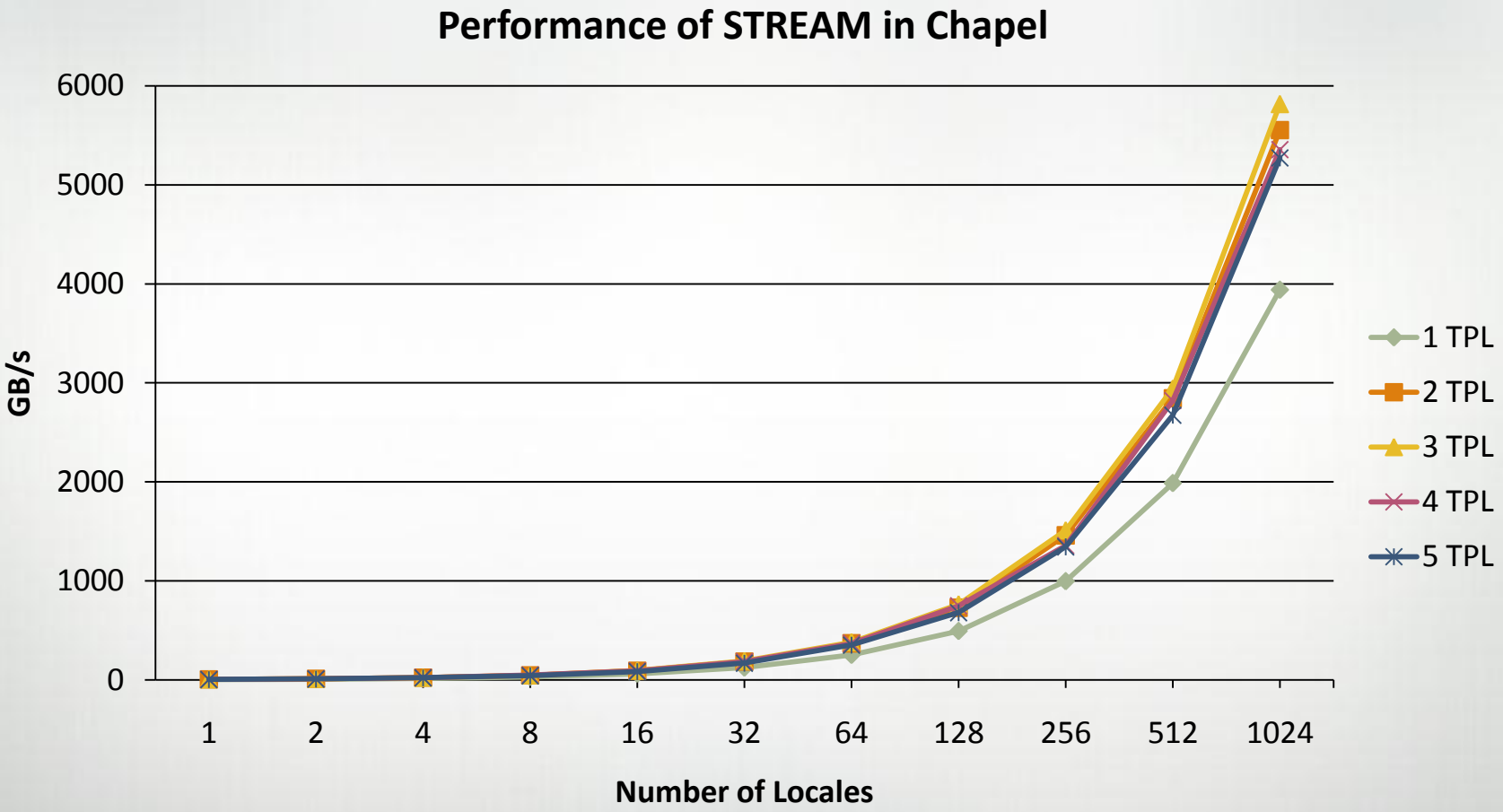
STREAM Triad in Chapel: Multi-Locale

Given: *m*-element vectors *A*, *B*, *C*

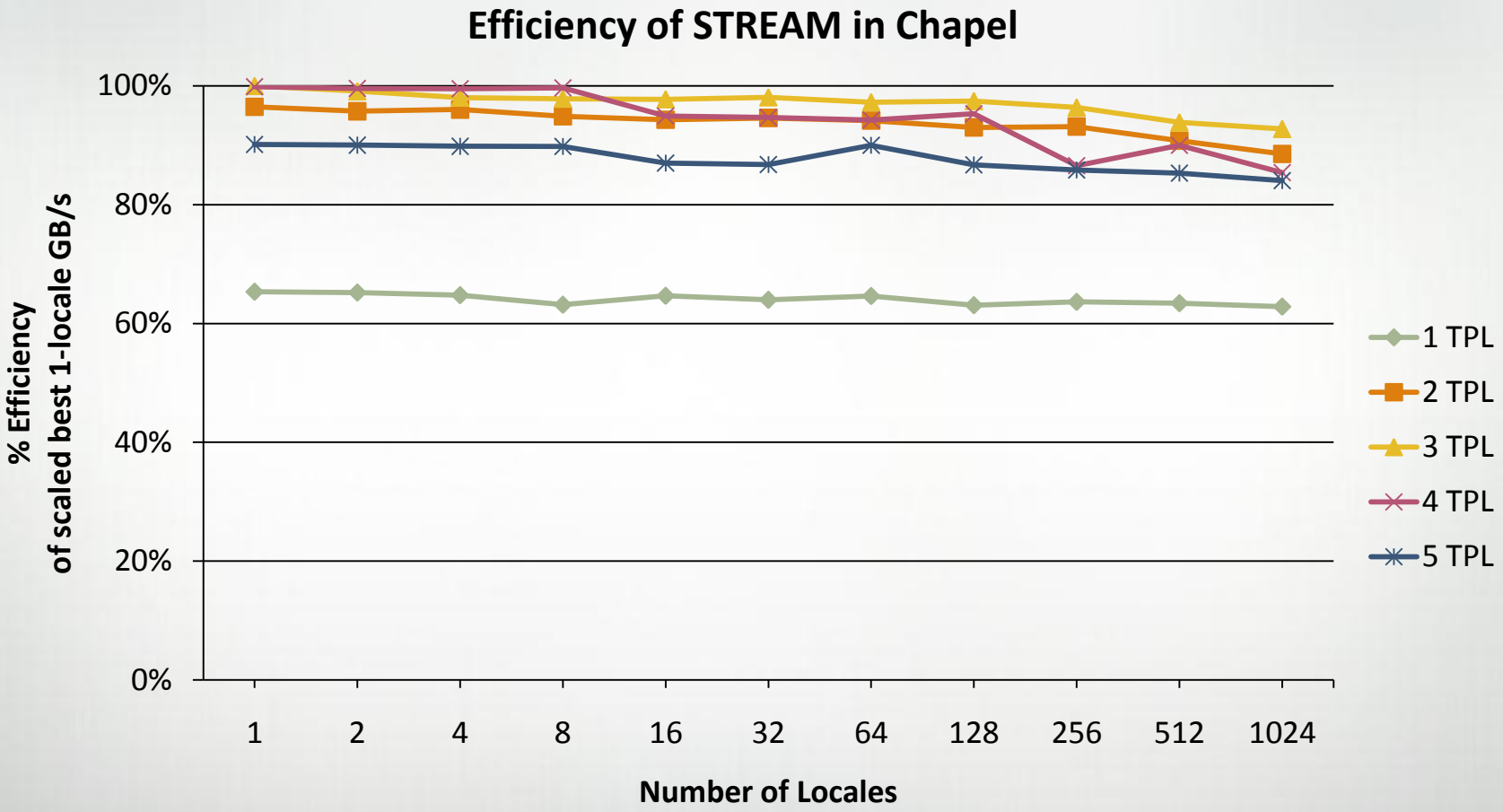
Compute: `forall i in 1..m do A(i) = B(i) + α * C(i);`

```
config const m: int(64) = ..., tpl = ...;
const alpha: real = 3.0;
const BlockDist = new Block(1,int(64),[1..m],tpl);
const ProblemSpace: domain(1, int(64))
                    distributed BlockDist = [1..m];
var A, B, C: [ProblemSpace] real;

forall (a,b,c) in (A,B,C) do
  a = b + alpha * c;
```

HPCC STREAM Performance



HPCC STREAM Efficiency



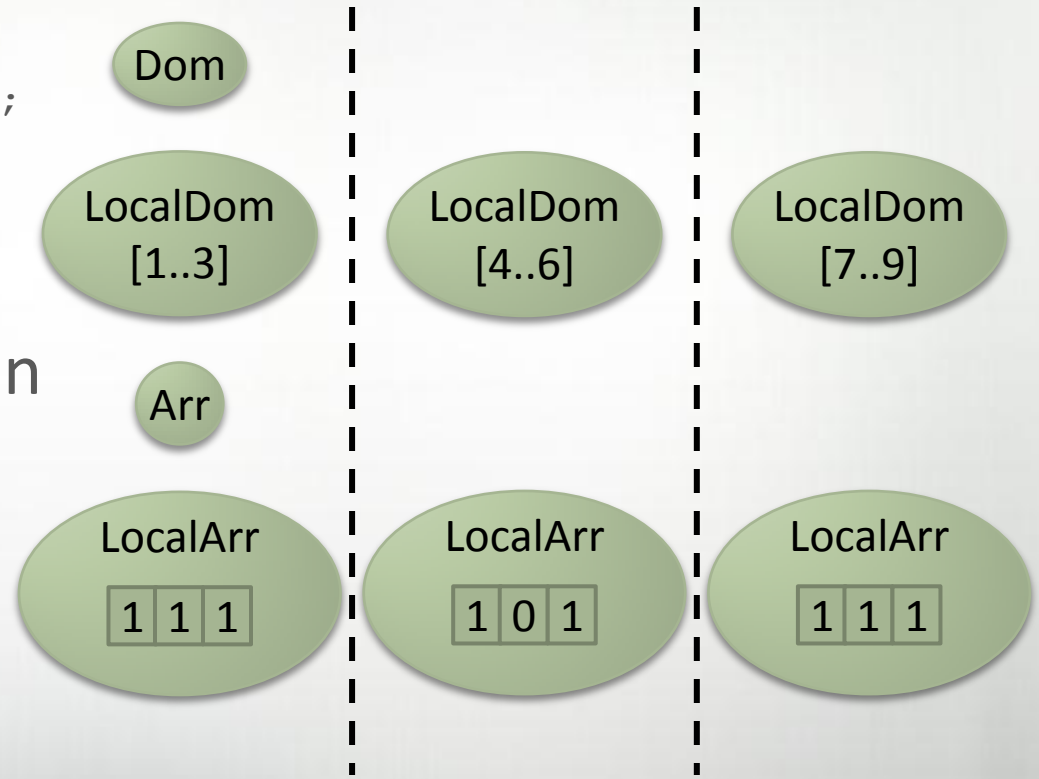
Optimization: Privatization

Simple example

```
var Dist: Block(1,int(64));
var Dom: domain(1,int(64))
           distributed Dist;
var Arr: [Dom] int;
```

Reference to local data requires communication

```
on Locales(1) {
  Arr(5) = 0;
}
```



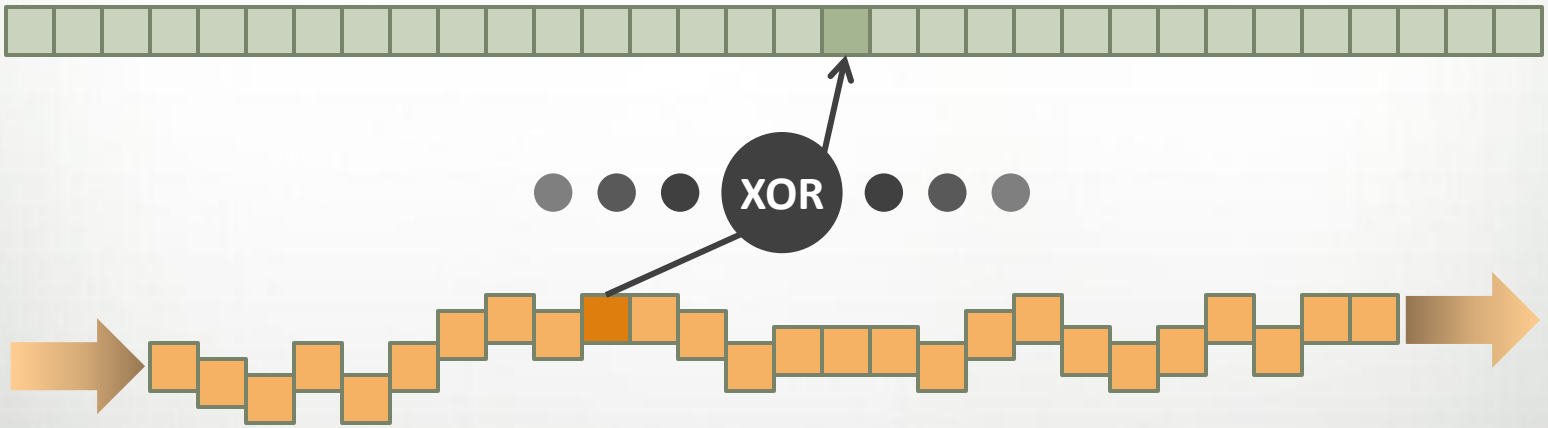
Outline

- HPCC STREAM Triad in Chapel
- HPCC RA in Chapel

Introduction to RA

Given: *m*-element table *T* (where $m = 2^n$)

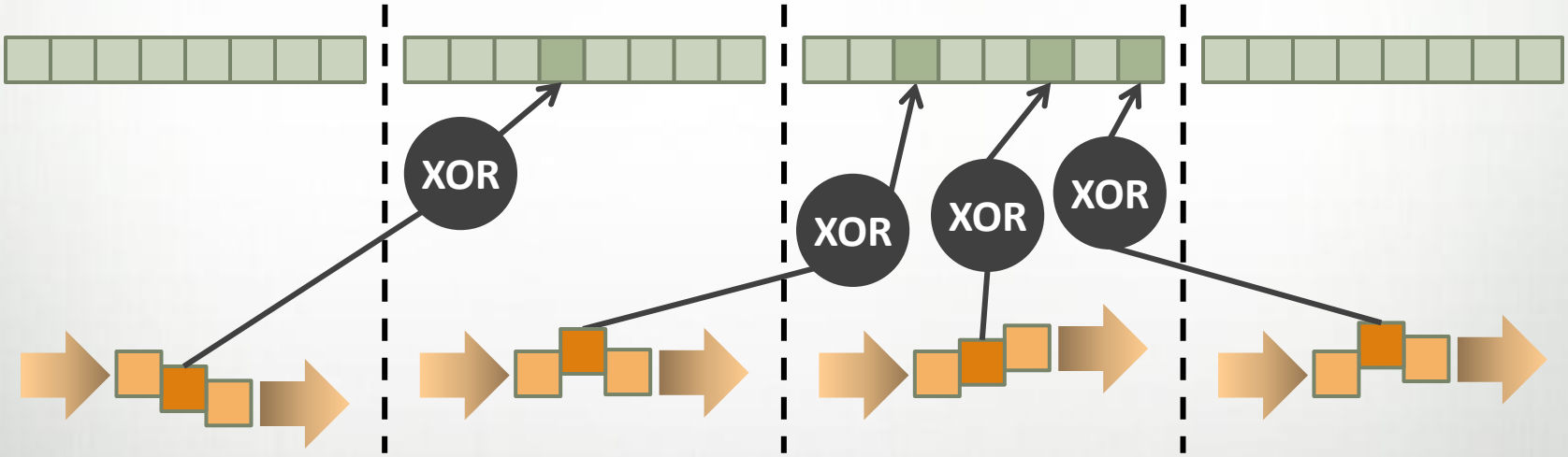
Compute:
```
forall r in RandomUpdates do
  T(r & (m-1)) ^= r;
```



Distributed Parallelization of RA

Given: *m*-element table *T* (where $m = 2^n$)

Compute:
```
forall r in RandomUpdates do
  T(r & (m-1)) ^= r;
```



RA in Chapel: Single Locale

Given: *m*-element table *T* (where $m = 2^n$)

Compute:
```
forall r in RandomUpdates do
  T(r & (m-1)) ^= r;
```

```
config const m = ..., N_U = ...;
const TableSpace: domain(1,uint(64)) = [0..m-1],
      Updates: domain(1,uint(64)) = [0..N_U-1];
var T: [TableSpace] uint(64);

forall (i,r) in (Updates,RAStream()) do
  T(r & (m-1)) ^= r;
```

RA in Chapel: Multi-Locale

Given: *m*-element table *T* (where $m = 2^n$)

Compute:
```
forall r in RandomUpdates do
  T(r & (m-1)) ^= r;
```

```
config const m = ..., N_U = ..., tpl = ...;
const TableDist = new Block(1,uint(64),[0..m-1],tpl),
      UpdateDist = new Block(1,uint(64),[0..N_U-1],tpl),
      TableSpace: domain(1,uint(64))
                  distributed TableDist = [0..m-1],
      Updates: domain(1,uint(64))
               distributed UpdateDist = [0..N_U-1];
var T: [TableSpace] uint(64);

forall (i,r) in (Updates,RAStream()) do
  on T(r & (m-1)) do
    T(r & (m-1)) ^= r;
```

RA in Chapel: Multi-Locale

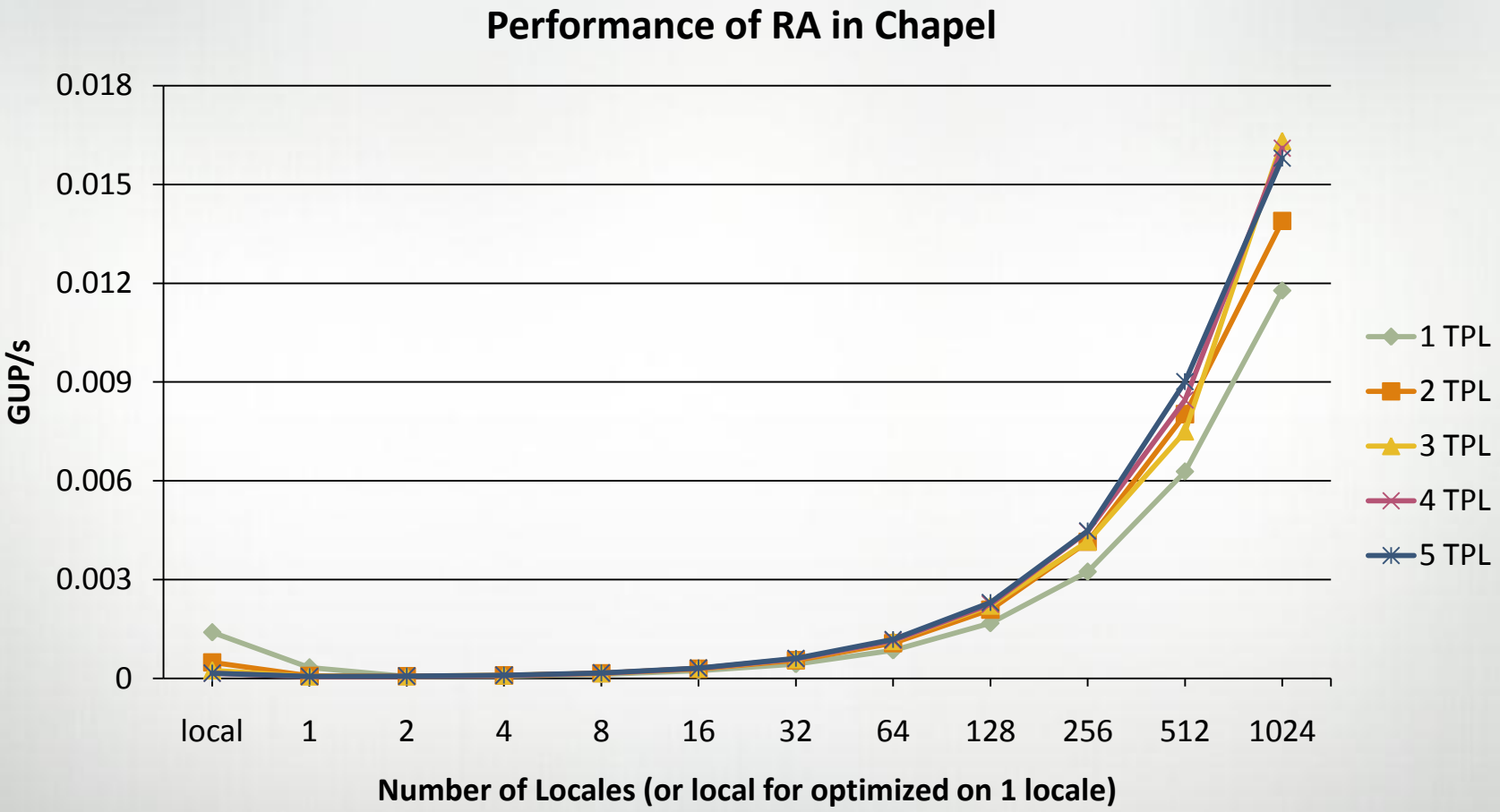
Given: *m*-element table *T* (where $m = 2^n$)

Compute: `forall r in RandomUpdates do T(r & (m-1)) ^= r;`

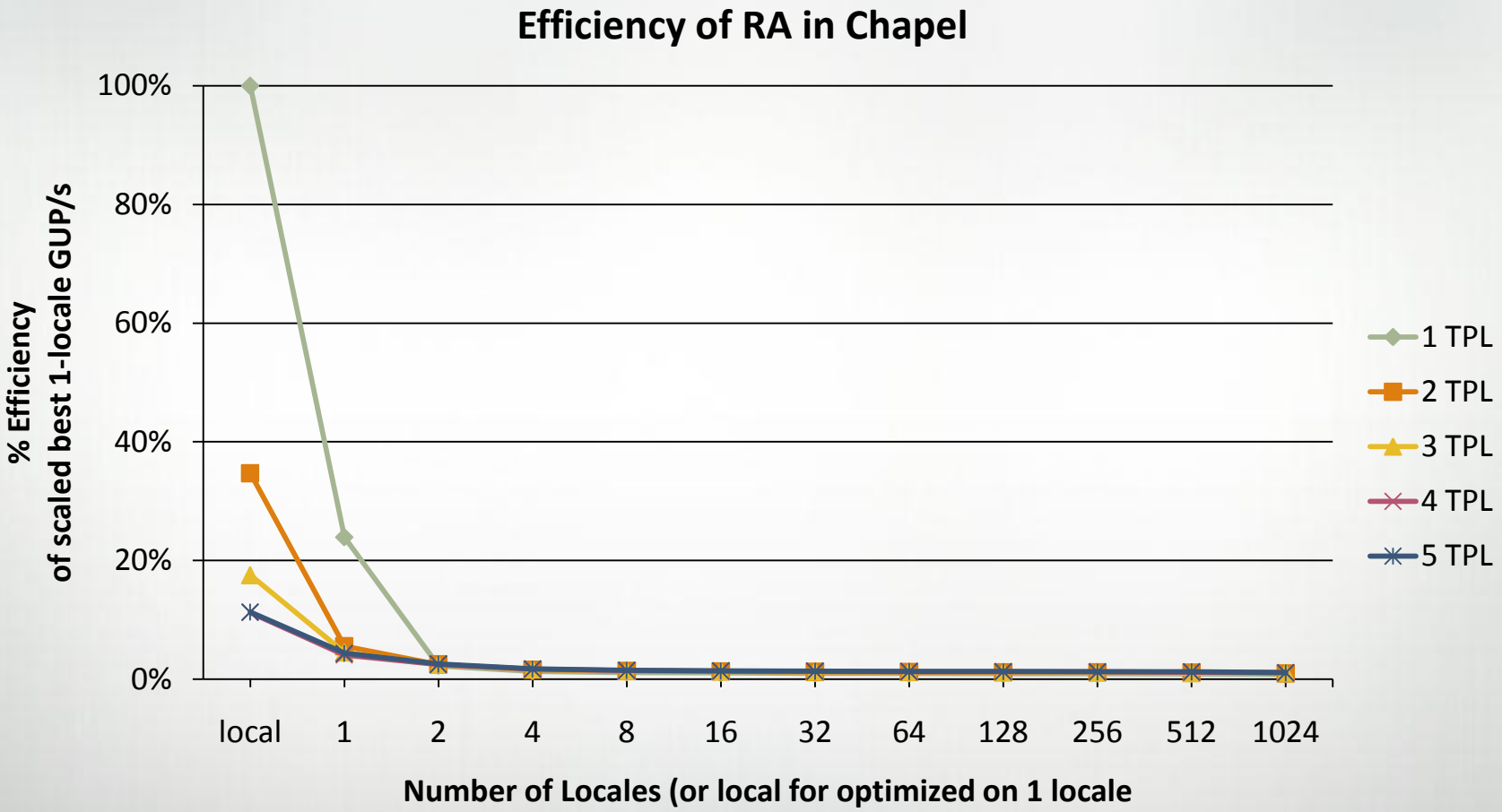
```
config const m = ..., N_U = ..., tpl = ...;
const TableDist = new Block(1,uint(64),[0..m-1],tpl),
      UpdateDist = new Block(1,uint(64),[0..N_U-1],tpl),
      TableSpace: domain(1,uint(64))
                    distributed TableDist = [0..m-1],
      Updates: domain(1,uint(64))
                   distributed UpdateDist = [0..N_U-1];
var T: [TableSpace] uint(64);

forall (i,r) in (Updates,RAStream()) do
  on T.domain.dist.ind2loc(r & (m-1)) do
    T(r & (m-1)) ^= r;
```

Call ind2loc method directly

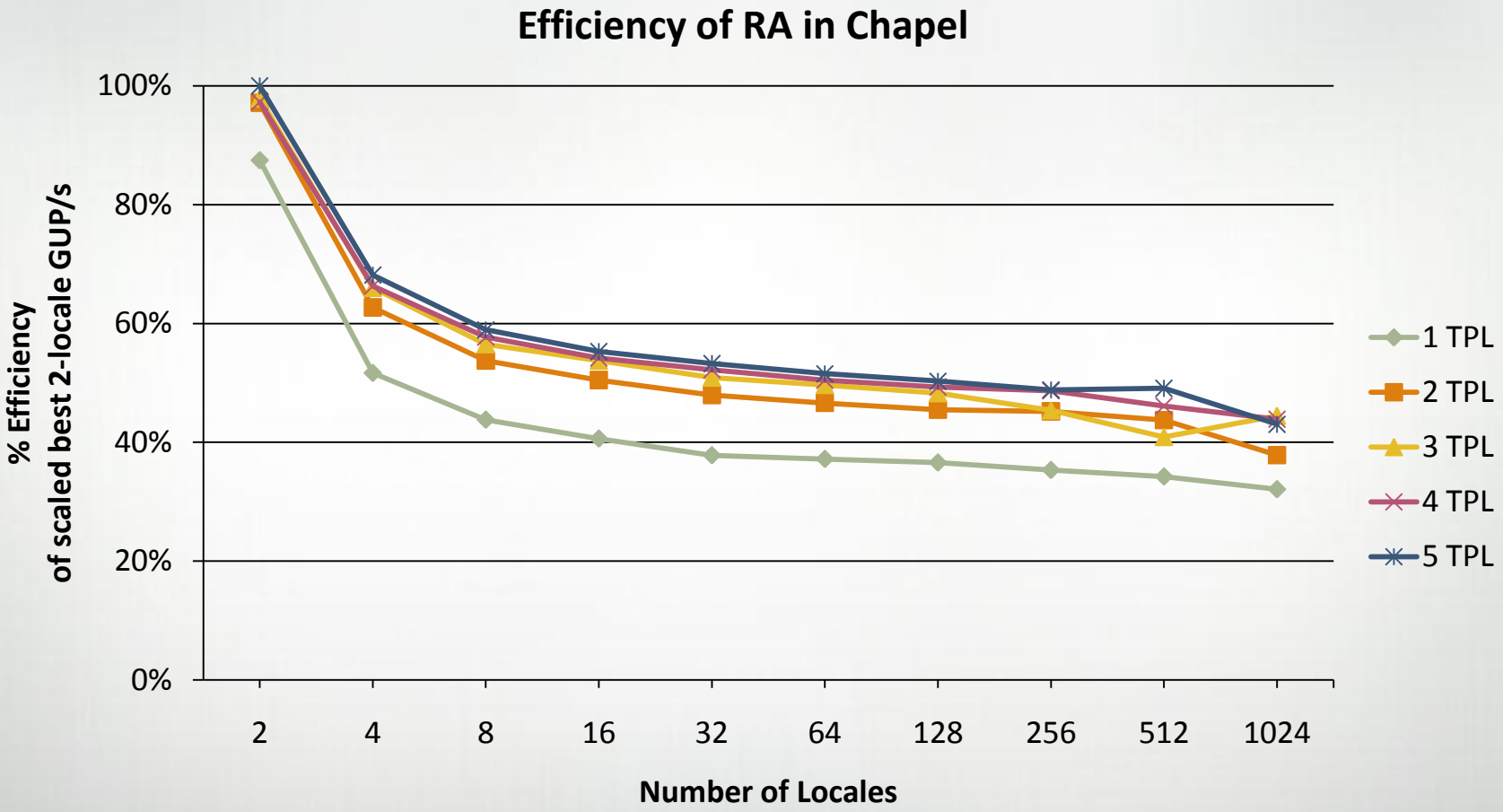
HPCC RA Performance



HPCC RA Efficiency I



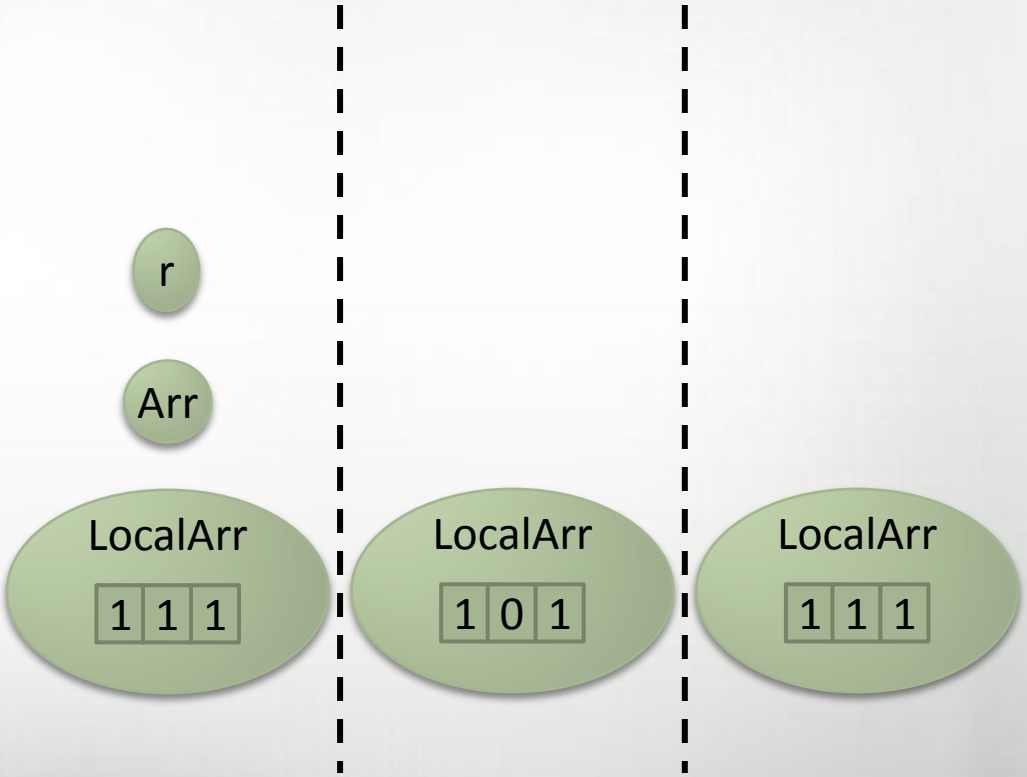
HPCC RA Efficiency II



Optimization: Remote Value Forwarding

Simple example

```
var Arr: [Dom] int;
var r: int;
on Locales(1) {
  Arr(r) ^= r;
}
```



Questions?

- HPCC STREAM Triad in Chapel
- HPCC RA in Chapel