Incomplete Models for Parallel Programming

Aaron Becker, Pritish Jetley, Phil Miller
7th Charm Workshop, 2009

PARALLEL
PROGRAMMING LAB
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS
PPL
UIUC

One Model to Rule Them All?

- multicore
- cluster
- GPU
- Cell
- many core
- irregular communication

- deadlock and race condition avoidance
- memory consistency
- transparent performance model
- critical path detection
- simple semantics

Incomplete Models

- Sometimes you don’t need full generality
- Complete freedom implies the freedom to create all possible incorrect programs

Incomplete Models

Downside

some things will not be expressible

Incomplete Models

Upside

some things will not be expressible

Incomplete Models

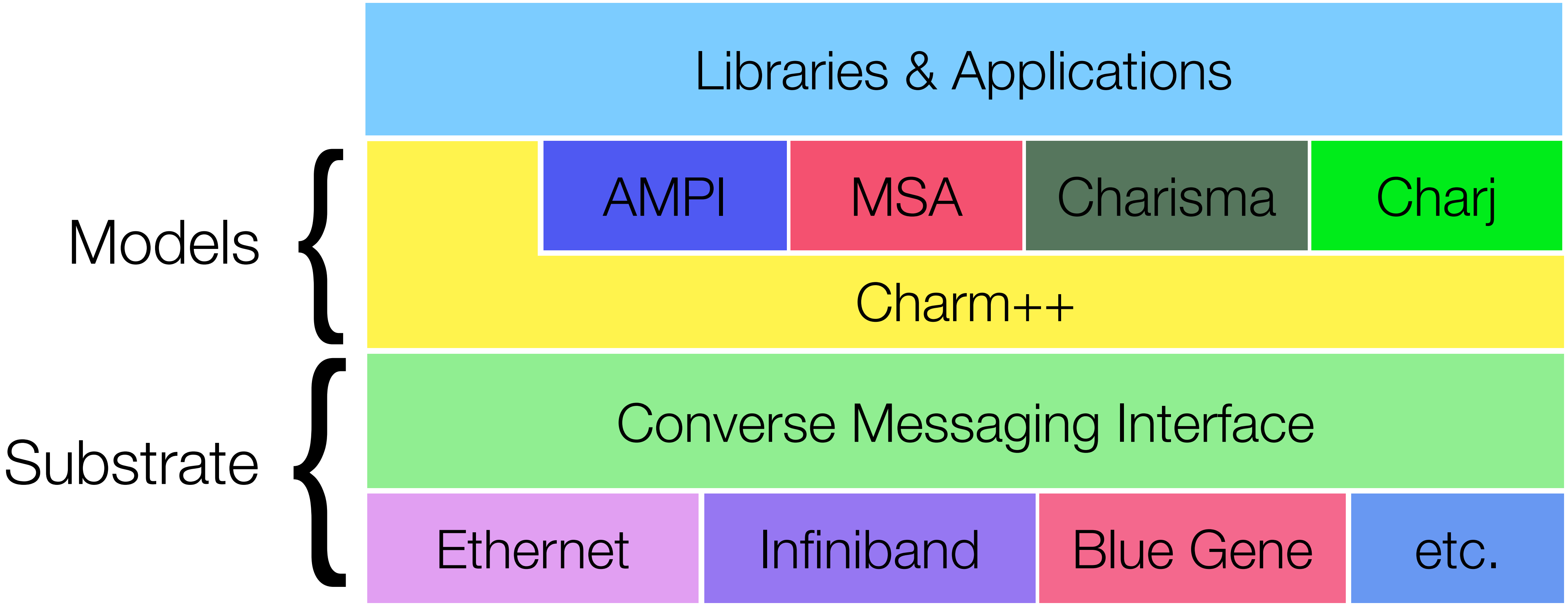
Upside

better safety guarantees

easier to get high performance

possibility for greater expressiveness

Common Infrastructure



Libraries & Applications

Models

AMPI

MSA

Charisma

Charj

Charm++

Substrate

Converse Messaging Interface

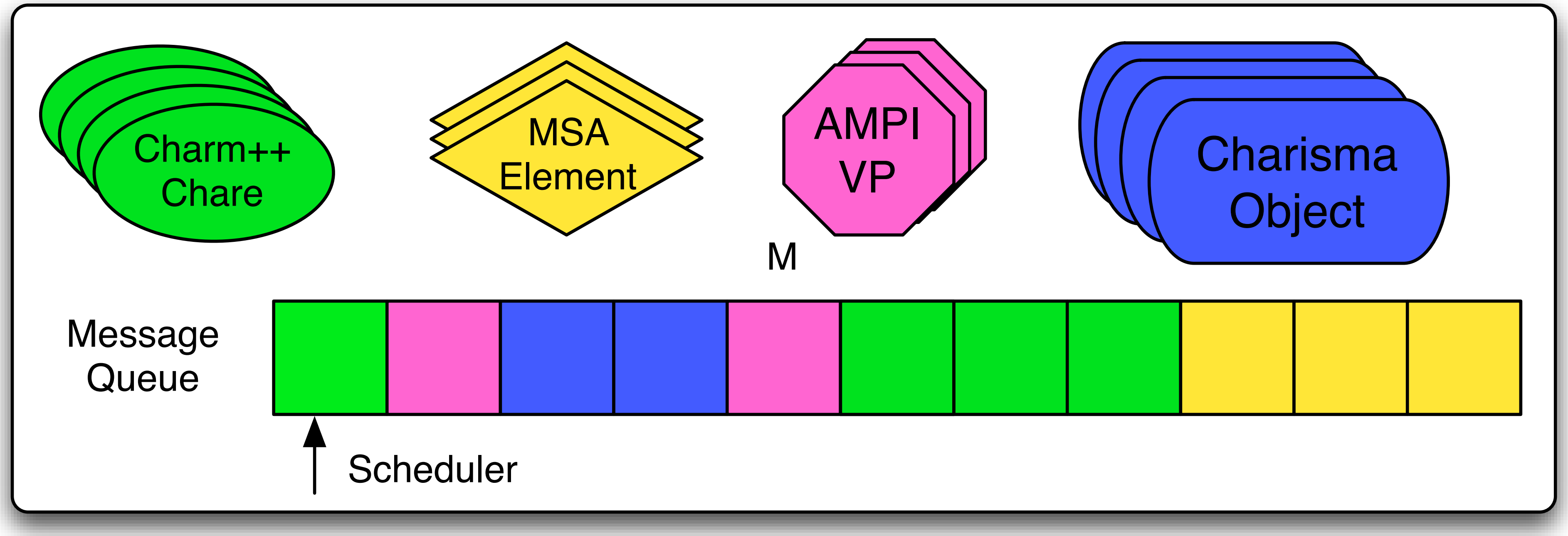
Ethernet

Infiniband

Blue Gene

etc.

Composition



Charj

- Glue language for incomplete models
- Has model knowledge which C++ compiler lacks
- Facilitates clean interaction between different models

Multiphase Shared Arrays

Discipline

- *Phase* defines allowed access
- Had phase before X10 existed
- Access modes for consistency seen in early Charm, Munin DSM

Modes

- Read-only: duh
- Write-once: each element may be written by one object/thread
- Accumulate: apply an associative, commutative operation (element-wise reduction)
- Owner-computes: some routine on local data

Benefit

- Adaptive fetching and caching
- No race conditions

Usage Flow

- Create
  - Dimension
  - Type
  - Shape
  - (Data Distribution)

Usage Flow

- Create
- Distribute
  - Send handle to all interested objects
- Sync to initialization mode
  - Parallel I/O with coordinates: Write-once
  - Generated based on coordinates: Owner-computes
  - Summed from other data: Accumulate
- Initialize

Usage Flow

- Create
- Distribute
- Sync to initialization mode
- Initialize
- Sync; Use; Sync; Use

Open questions

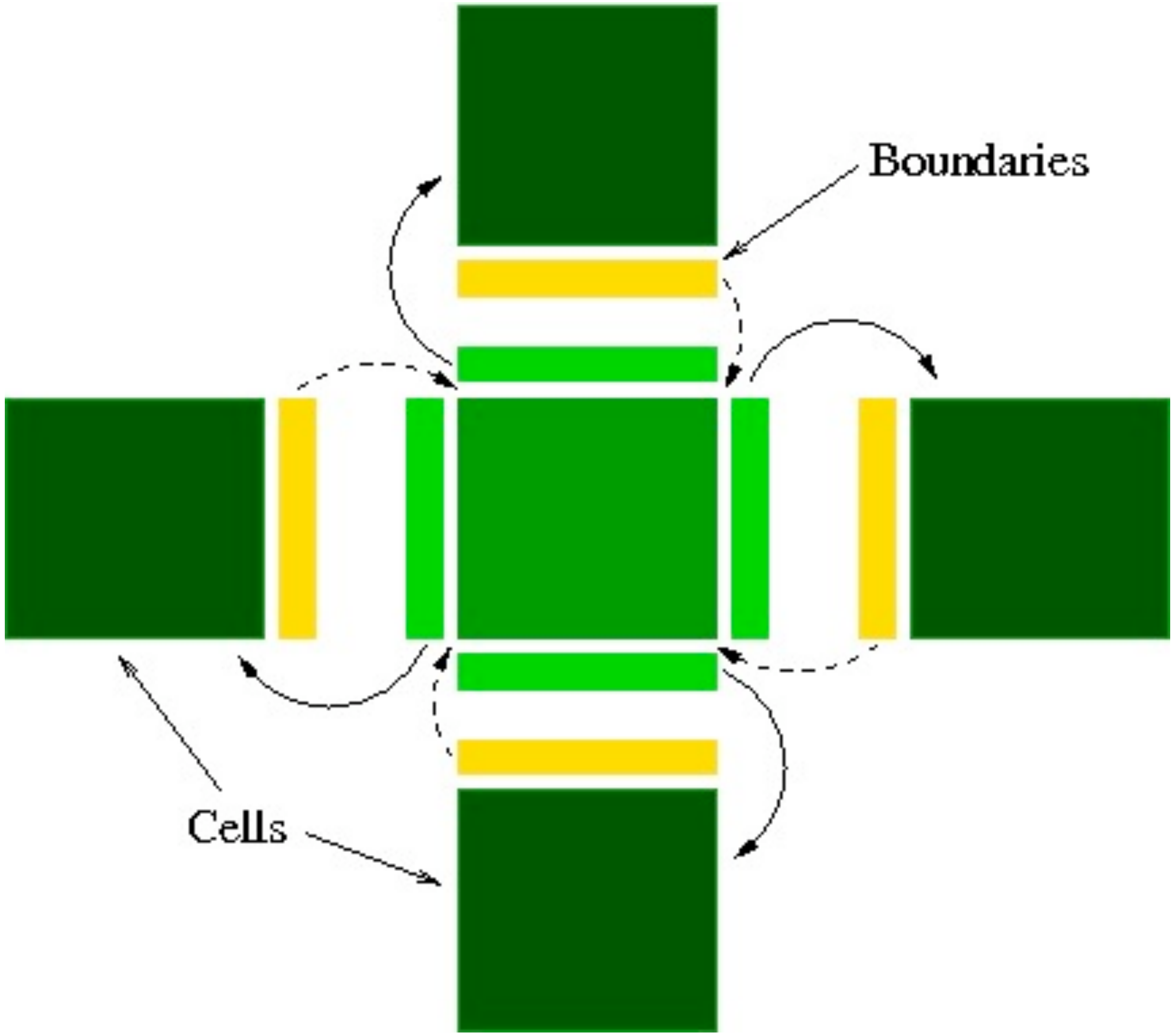
- How much can the compiler enforce?
- How to match distribution to application?
- What other modes make sense?

Charisma

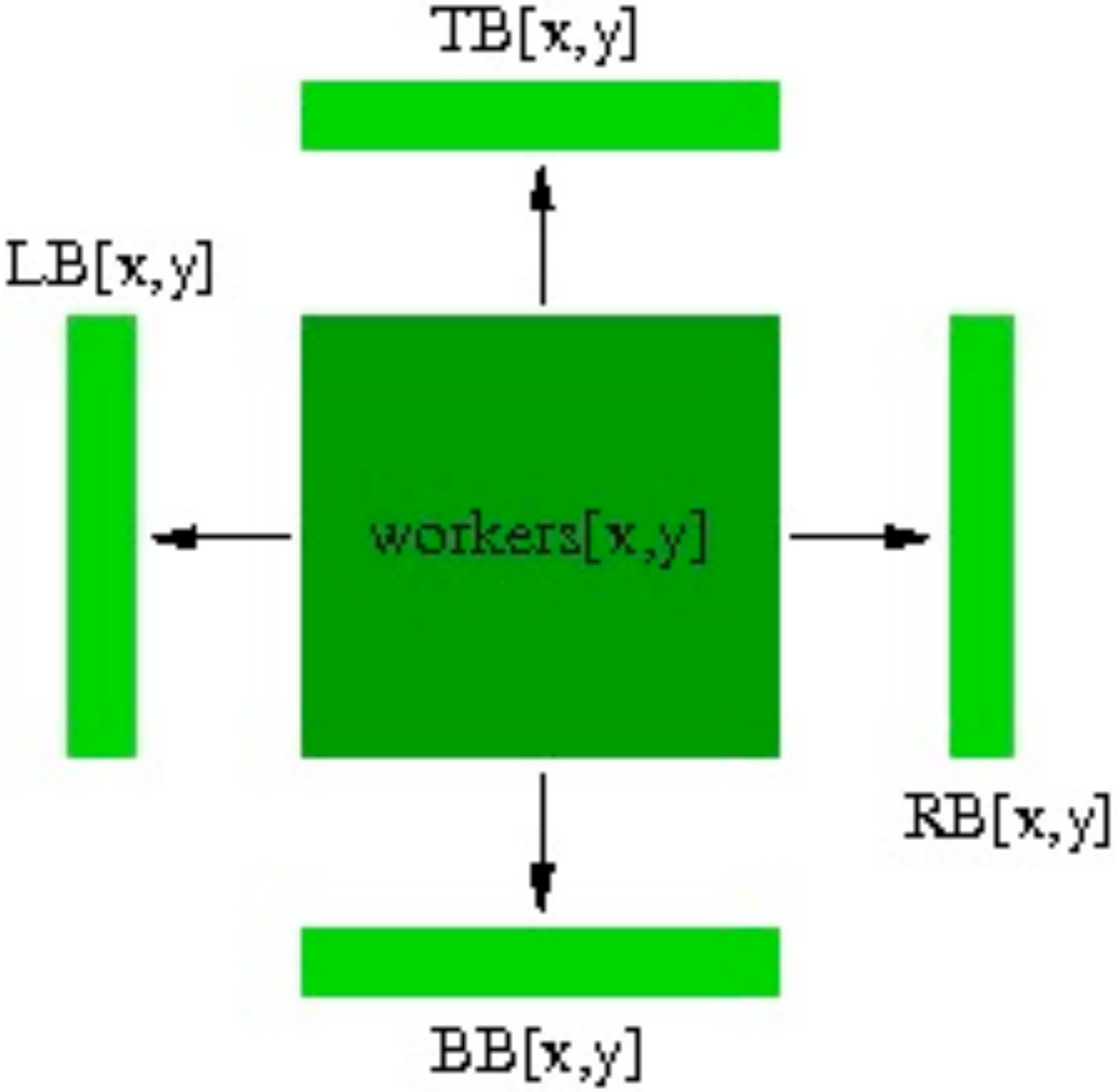
Static Dataflow

- Objects communicate with fixed sets of neighbors
- *Produce* and *consume* parameters
- Defines a powerful paradigm on which several classes of applications can be based
  - Structured meshes
  - Unstructured meshes
  - Molecular dynamics

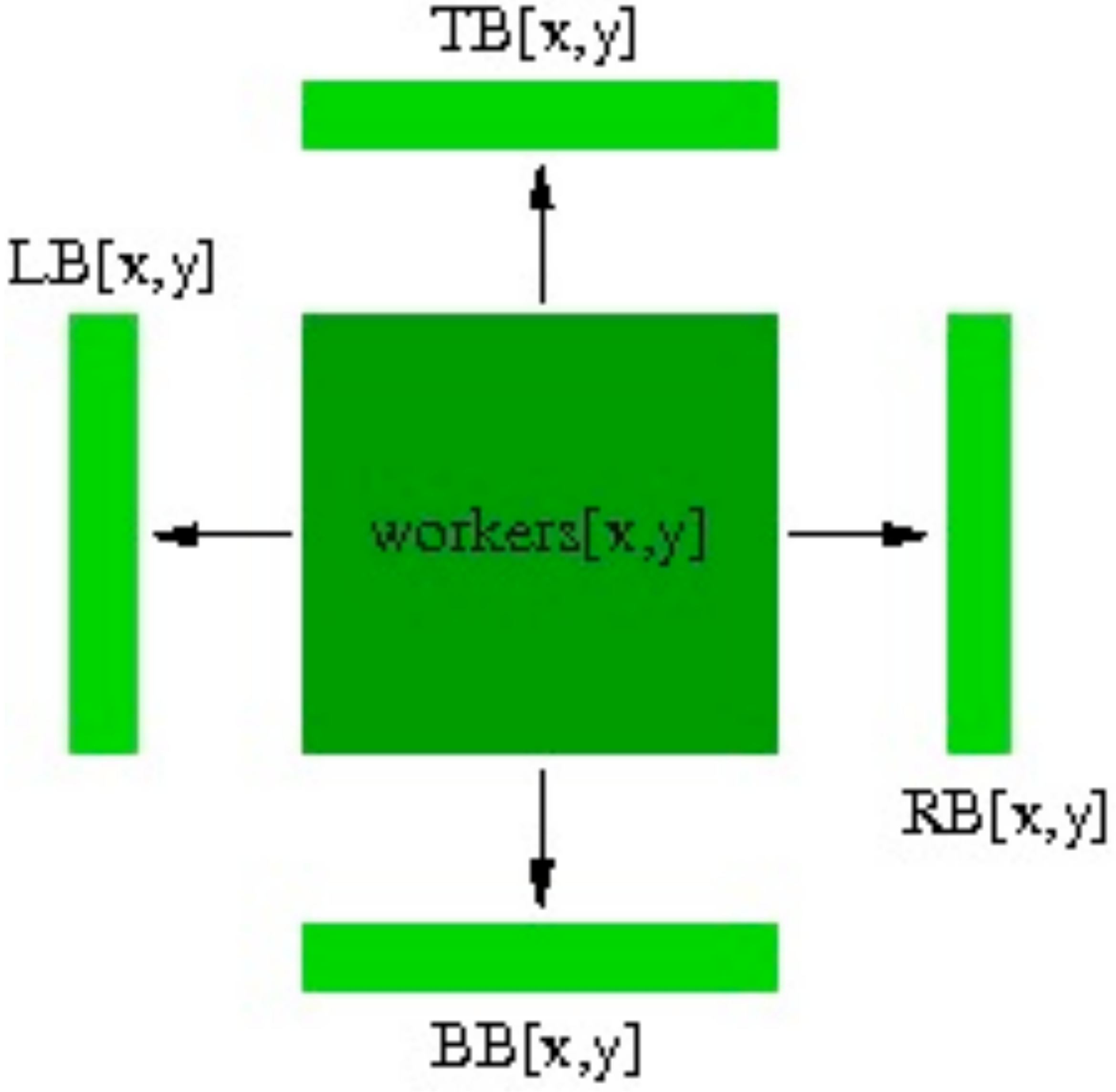
Stencil computation



Produce own boundaries

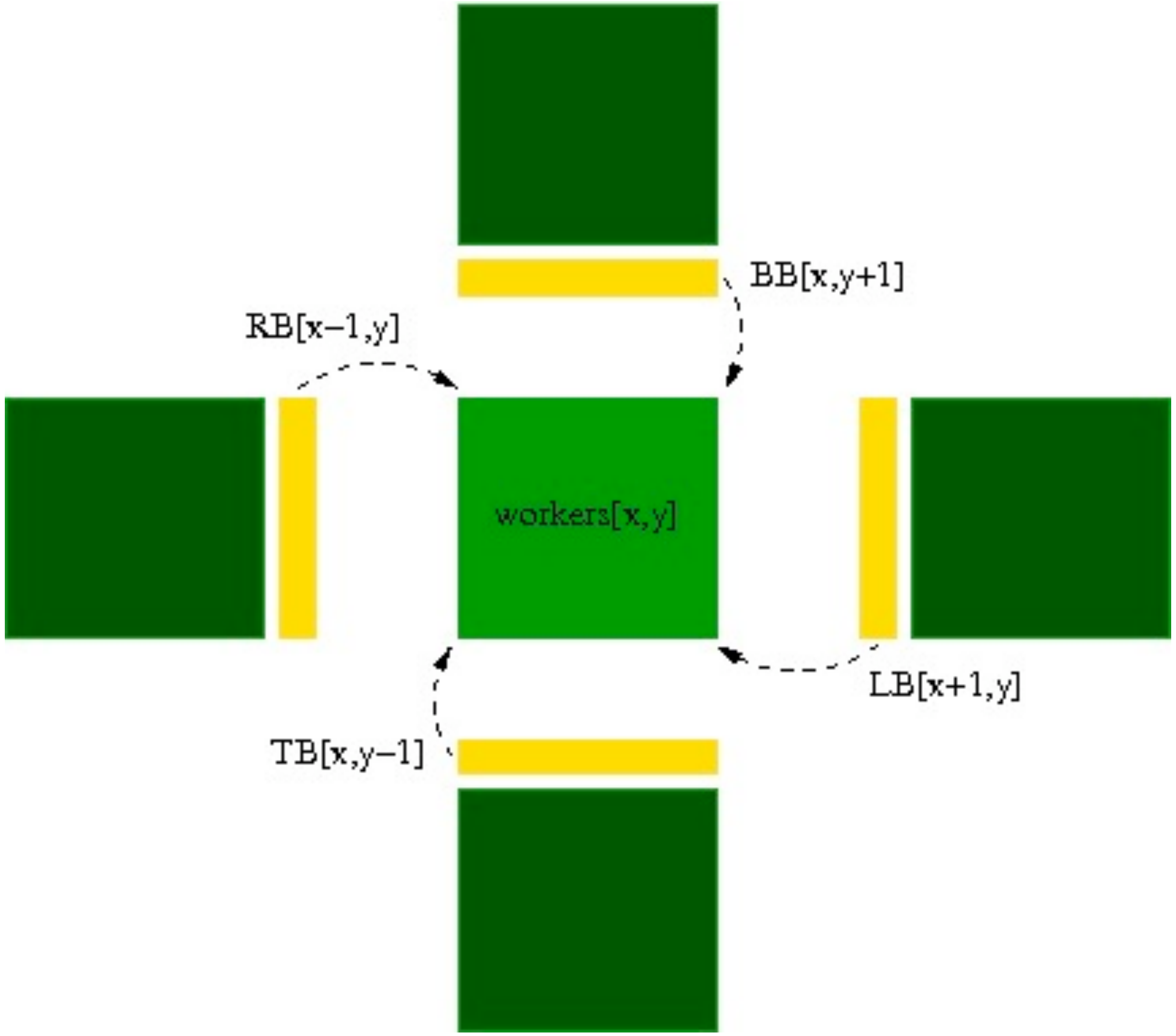


Produce own boundaries

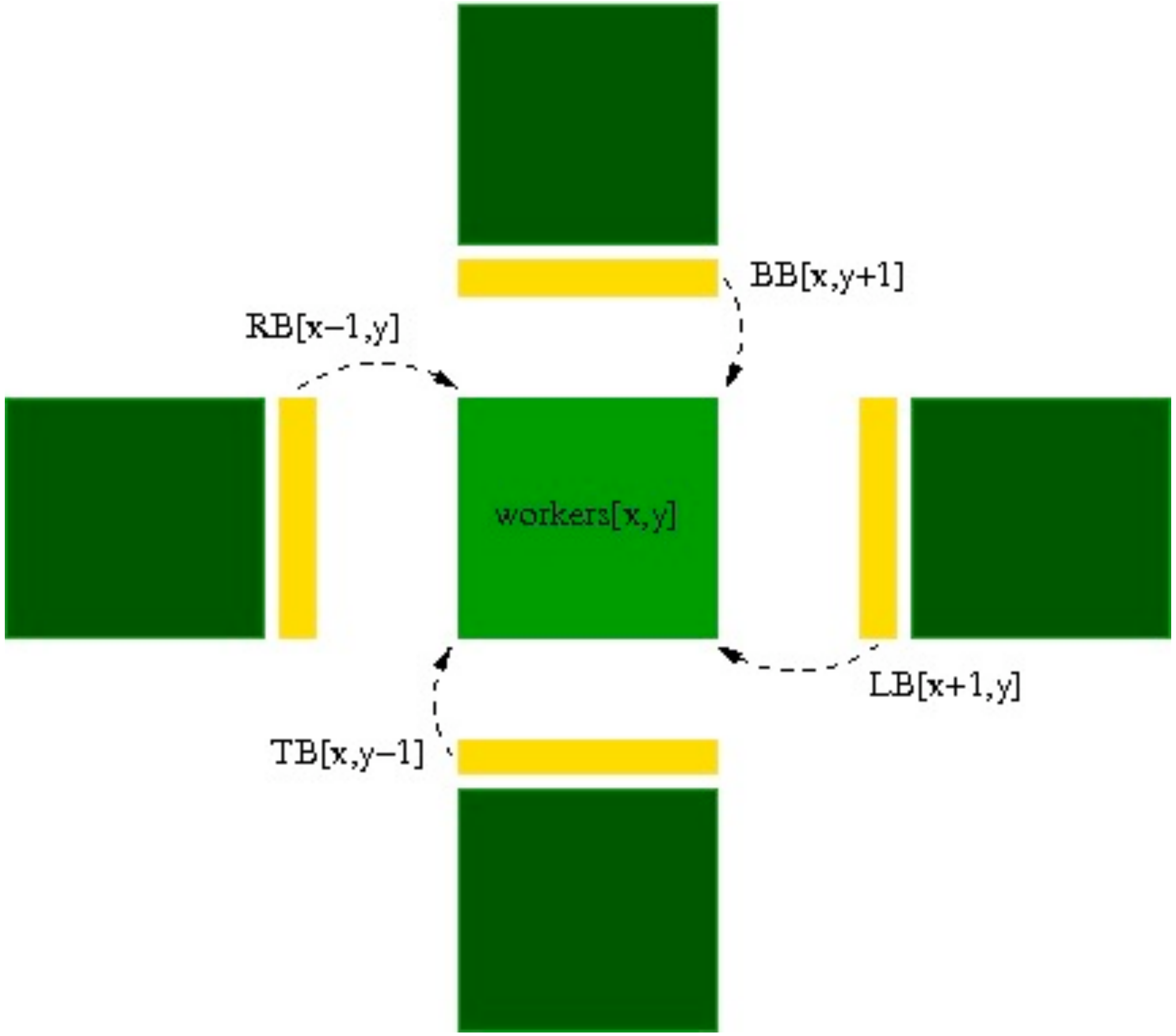


```
foreach x,y in workers
  (LB[x,y],RB[x,y],
   TB[x,y],BB[x,y])
   <- workers[x,y].prod()
end-foreach
```

Consume neighbors' boundaries



Consume neighbors' boundaries



```
foreach x,y in workers
   (+err)
    <- workers[x,y].cons(
        LB[x+1,y],
        RB[x-1,y],
        TB[x,y-1],
        BB[x,y+1])↰
end-foreach
```

Communication patterns

- Point-to-point

```
(param[i]) ← obj[i].prod()↰
...
obj[i].cons(param[i-1])↰
```

Communication patterns

- Reduction

```
foreach x,y in cells
  (+err) ← cells[x,y].compute();
end-foreach
...
main.reportError(err);
```

Communication patterns

- Multicast

```
foreach x in A
  (points[x]) ← A[x].prod();
end-foreach
foreach x,y in B
  B[x,y].cons(points[x]);
end-foreach
```

Communication patterns

- Scatter

```
foreach x in A
  (points[x,*]) ← A[x].prod();
end-foreach
foreach x,y in B
  B[x,y].cons(points[x,y]);
end-foreach
```

Communication patterns

- Gather

```
foreach x,y in A
  (points[x,y]) ← A[x,y].prod();
end-foreach
foreach x in B
  B[x].cons(points[x,*]);
end-foreach
```

Future directions

- Bags of neighbors
    - Affine expressions can do only so much
    - e.g. unstructured meshes
- Topology mapping
    - Place communicating objects together
- Streaming extensions

Future directions

- Incremental gather
  - The Agarwal *et al.* algorithm for matrix multiplication:

```
foreach x,y,z in w
    (A[x,y,z]) <- workers[x,y,z].produceA();
    (B[x,y,z]) <- workers[x,y,z].produceB();
    (+C[x,z]) <- workers[x,y,z].mult(A[x,y,*],
                                    B[*,y,z]);
end-foreach
```

  - `mult` can be performed in a piecemeal fashion; more overlap of comp/comm.