

# Is MPI Suitable for a Generative Design-Pattern System?

Paras Mehta, Nelson Amaral, and  
Duane Szafron

May 6, 2005



# Outline

---

- **C**orrect **O**bject-**O**riented **P**attern-Based **P**arallel **P**rogramming **S**ystem (CO<sub>2</sub>P<sub>3</sub>S)
- **M**PI/C **A**dvanced **P**attern-Based **P**arallel **P**rogramming **S**ystem (MAP<sub>3</sub>S)
  - Comparison with CO<sub>2</sub>P<sub>3</sub>S
  - Patterns
  - Results
  - Future work





PROGRAM WINDOW

File Environment Program Help

Palette

Program

CO<sub>2</sub>P<sub>3</sub>S SolutionA 1744

Wavefront

Wavefront Editor: Class

3	2	1
4		

Dependencies

Full Matrix

Push

Flavor Type: Flavor

Neighbors Only: True

Computing Science

# Motivation: CO<sub>2</sub>P<sub>3</sub>S

---

- **C**orrect **O**bject-**O**riented **P**attern-Based **P**arallel **P**rogramming **S**ystem
  - 1) select a pattern
  - 2) set parameters
  - 3) generate framework
  - 4) insert application-specific code



# CO<sub>2</sub>P<sub>3</sub>S

---

- Patterns implemented in Java
  - Five shared-memory patterns using multi-threaded Java
  - Four distributed memory patterns using RMI and Jini



# CO<sub>2</sub>P<sub>3</sub>S

---

- Strengths

- object-oriented code lends well to frameworks
- “write once, run anywhere”
- user writes only sequential, application-specific code

- Shortcomings

- as a parallel programming system, Java not widely used
- performance issues



# MAP<sub>3</sub>S

---

- **M**PI/C **A**dvanced **P**attern-Based **P**arallel **P**rogramming **S**ystem
- Goals:
  - 1) Recreate CO<sub>2</sub>P<sub>3</sub>S' usability
  - 2) Appeal to broader user base
  - 3) Improve performance
  - 4) Improve user control of performance



Making  
IT  
happen

Computing Science

# Mesh Pattern

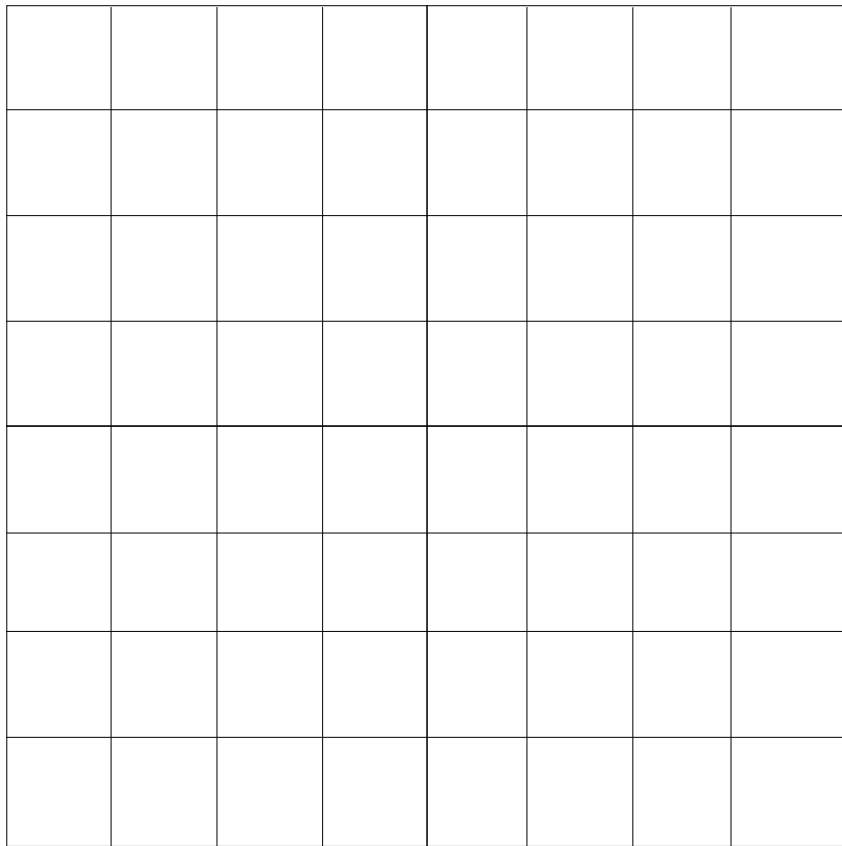
---

- Mesh: multi-dimensional arrangement of data elements
- Problems involving iteration over matrices
  - Conway's Game of Life
  - Morphogen Pair simulation
- Data parallel problems



# Mesh Pattern

---

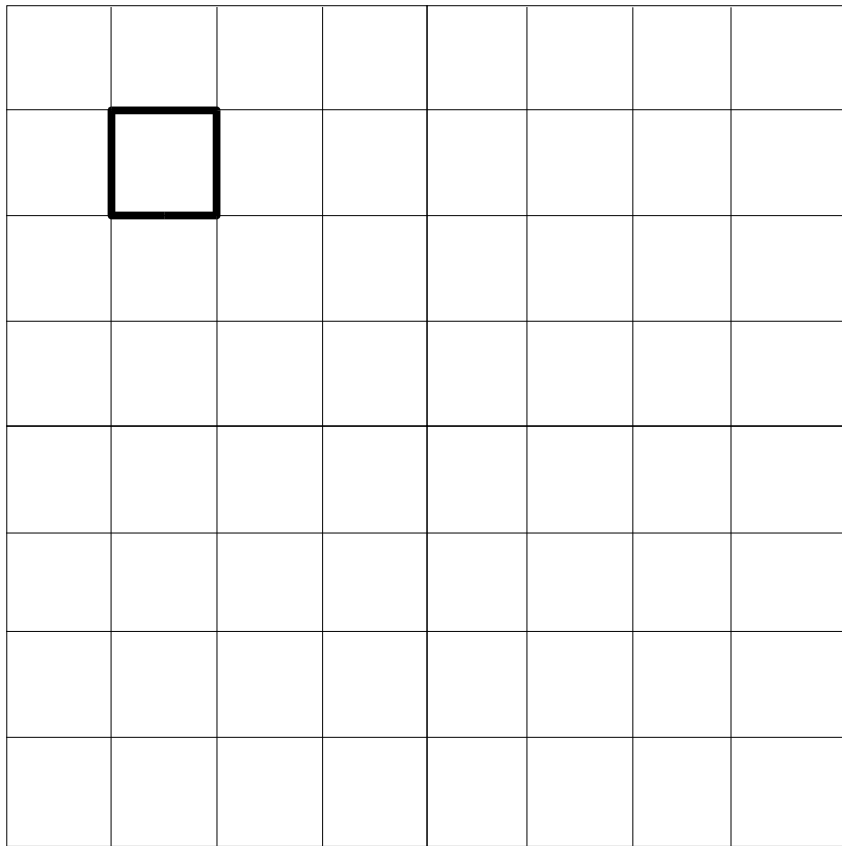


- Example
  - 8x8 mesh



# Mesh Pattern

---

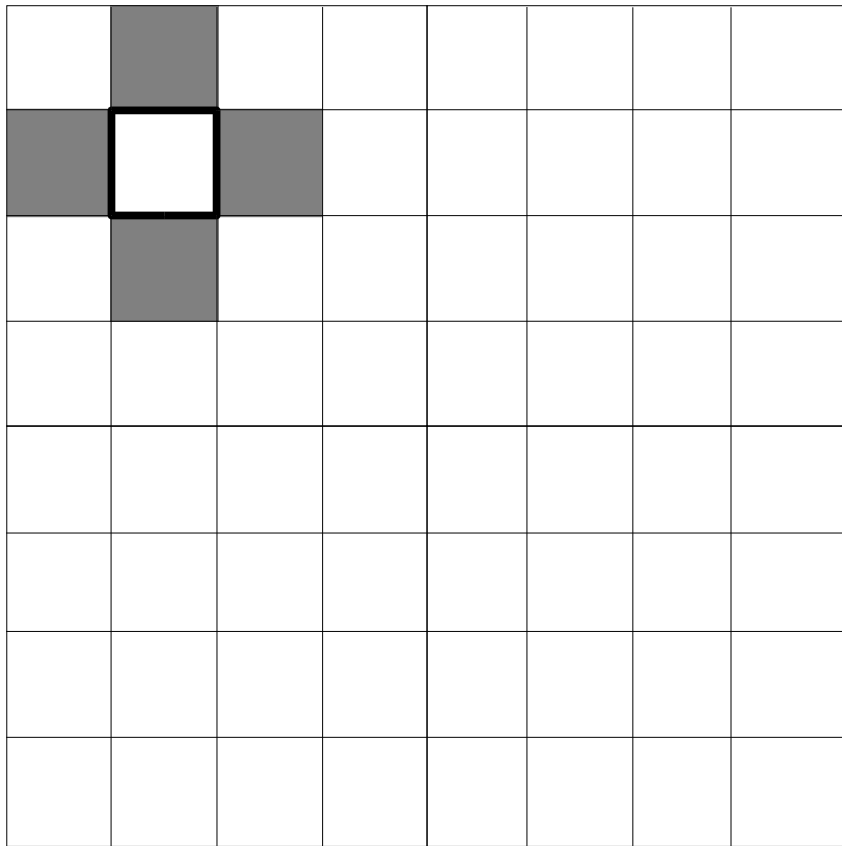


- Example
  - 8x8 mesh
  - Iterate over all elements



# Mesh Pattern

---

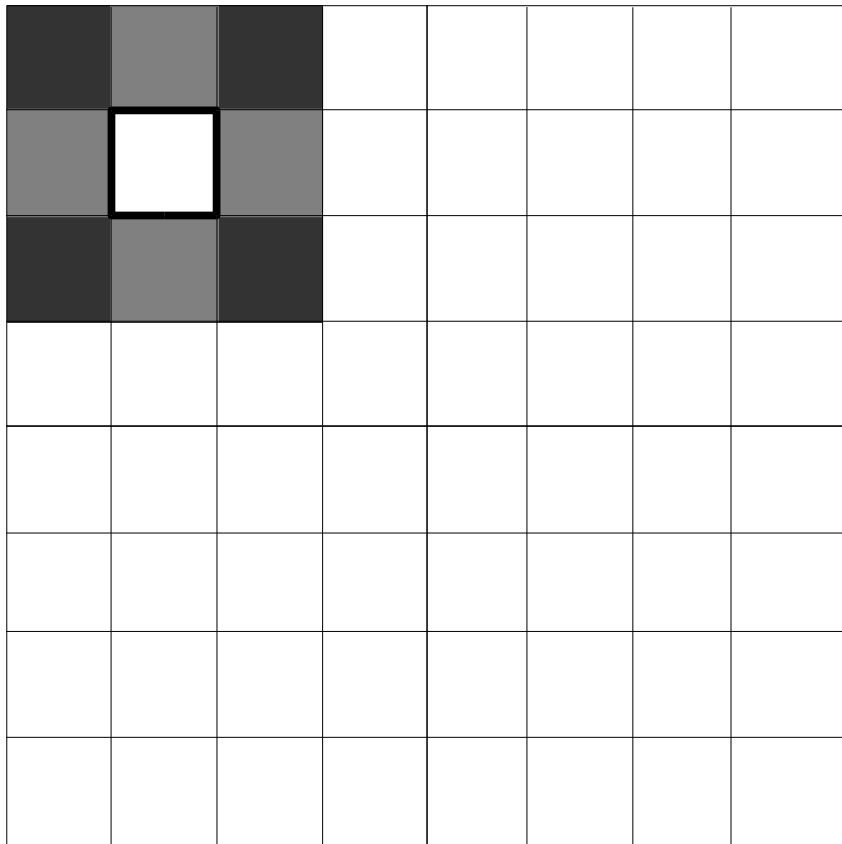


- Example
  - 8x8 mesh
  - Iterate over all elements
  - Next value depends on neighbours' values



# Mesh Pattern

---

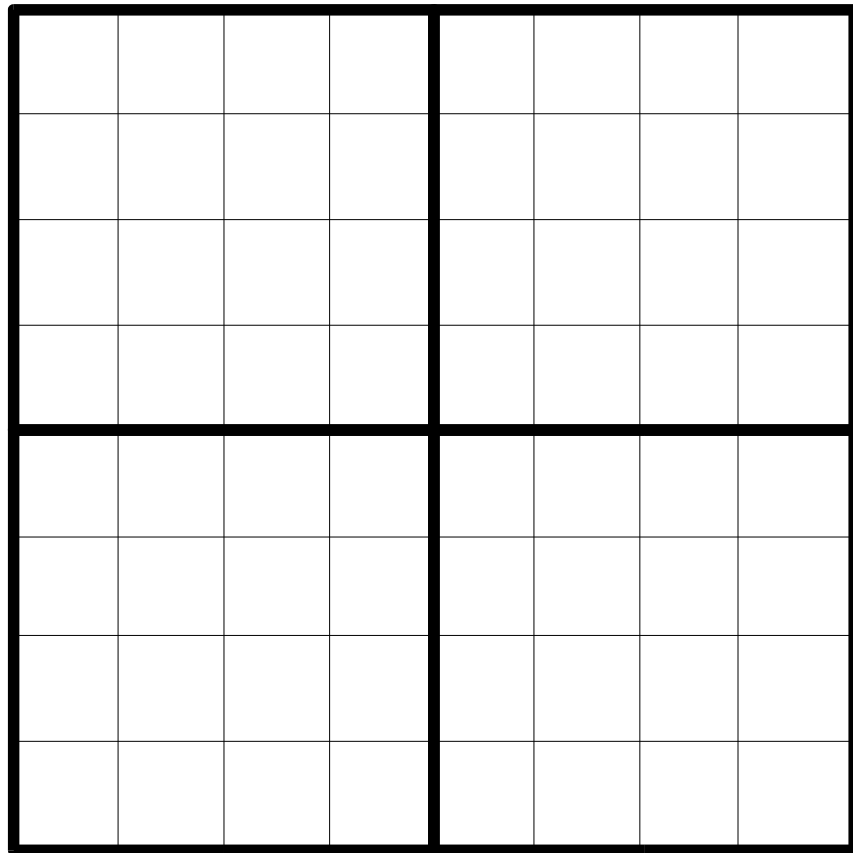


- Example
  - 8x8 mesh
  - Iterate over all elements
  - Next value depends on neighbours' values



# Mesh Pattern

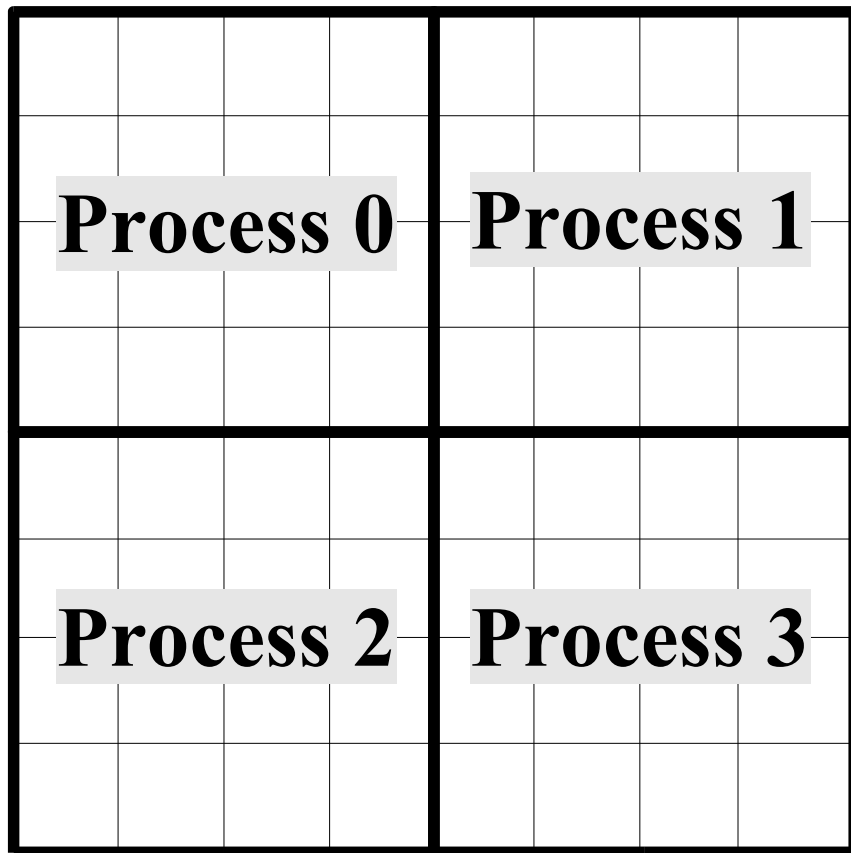
---



- Example
  - 8x8 mesh
  - Divide into 4x4 blocks



# Mesh Pattern

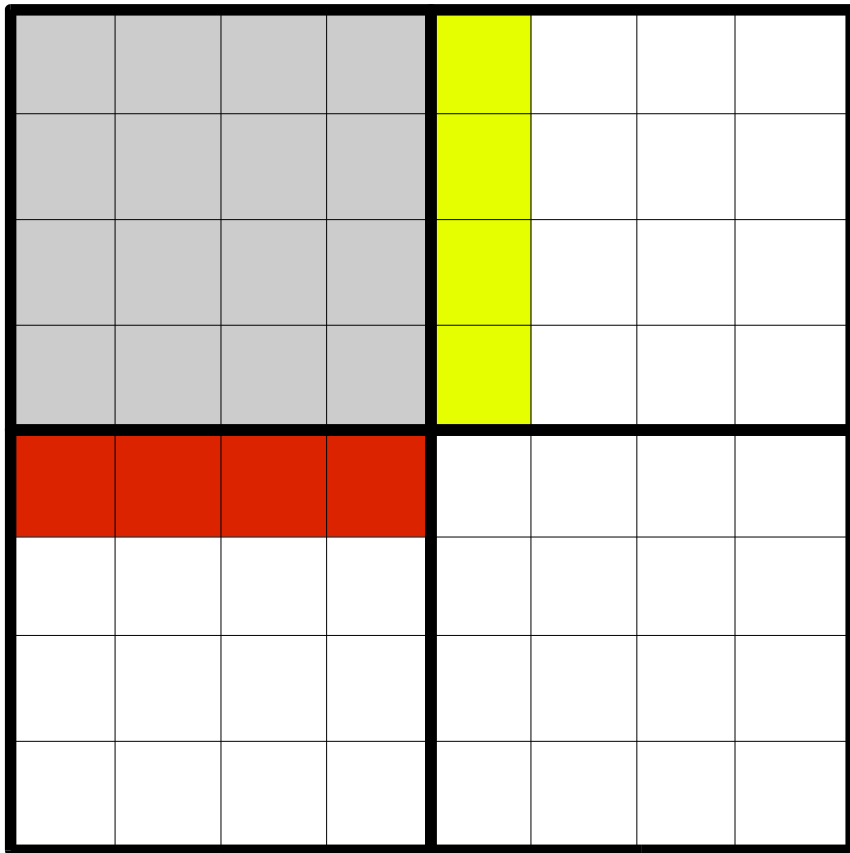


- Example
  - 8x8 mesh
  - Divide into 4x4 blocks
  - Distribute round robin to four processes



# Mesh Pattern

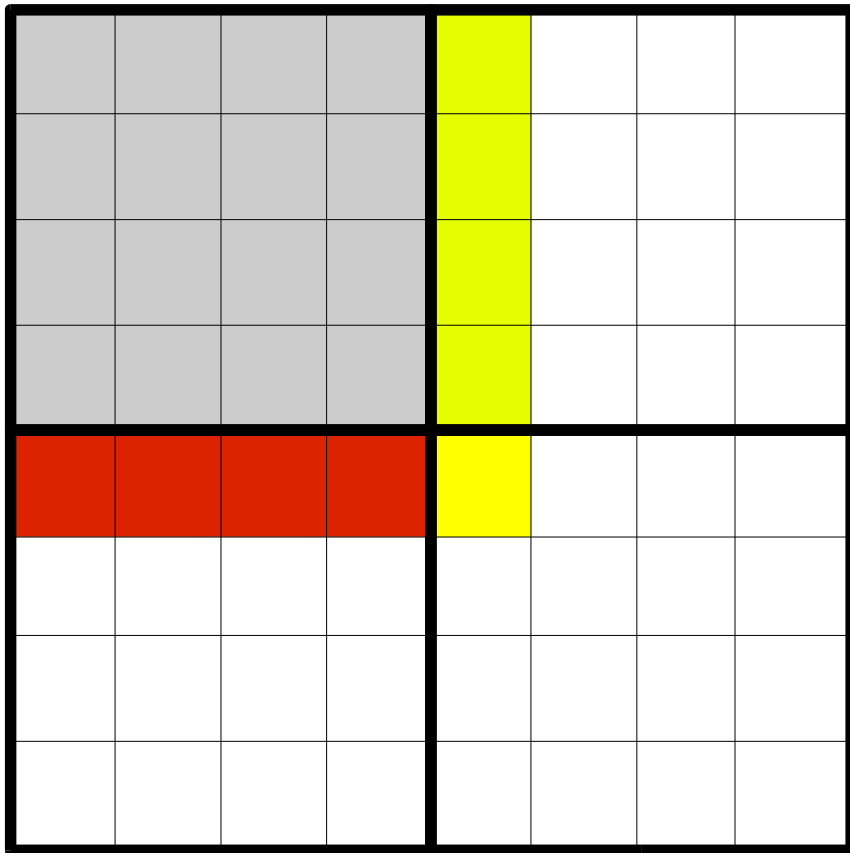
---



- Example
  - Need to synchronize at borders between iterations



# Mesh Pattern

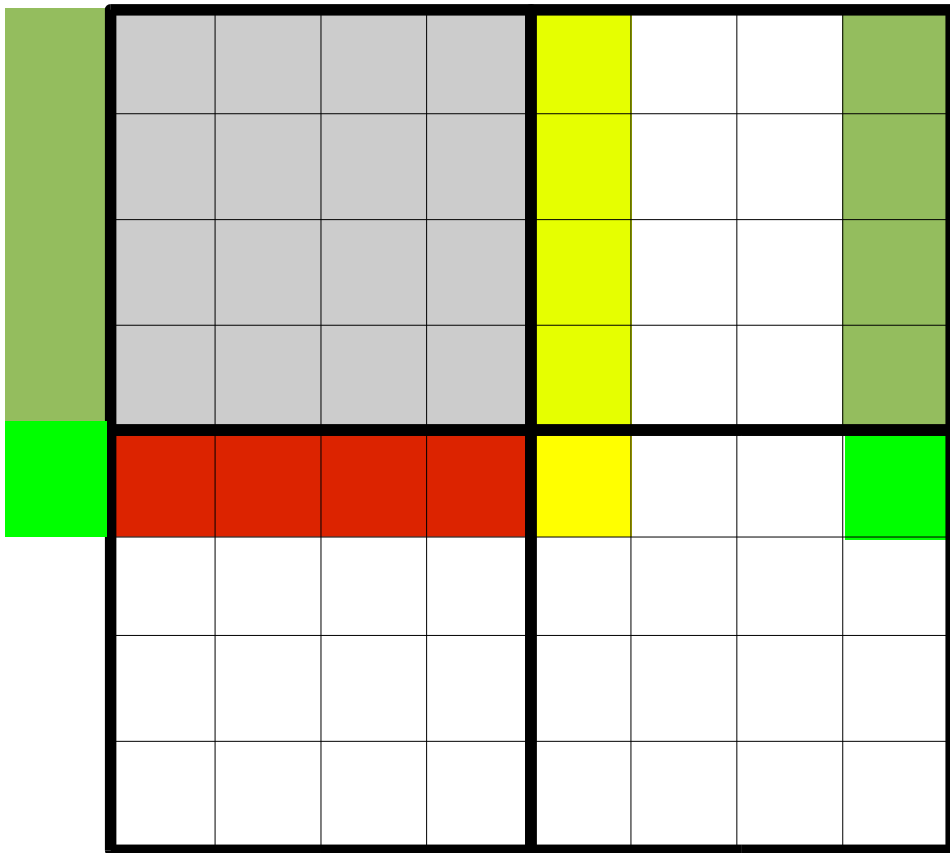


- Example
  - Need to synchronize at borders between iterations
  - Eight neighbours for each element requires more communication



# Mesh Pattern

---



- Example
  - Toroidality requires still more communication



# Mesh Pattern - CO<sub>2</sub>P<sub>3</sub>S

---

- CO<sub>2</sub>P<sub>3</sub>S
  - Parameters:
    - neighbours, toroidality, iteration type
  - Main Hooks:
    - iteration methods
    - termination condition
    - initialize
    - reduce



# Mesh Pattern - CO<sub>2</sub>P<sub>3</sub>S

---

- Main loop:

```
initialize();  
while(notDone()) {  
    prepare();  
    iterate();  
}  
reduce();
```



Making  
IT  
happen

Computing Science

# Mesh Pattern - CO<sub>2</sub>P<sub>3</sub>S

---

- Main loop:

```
initialize();  
while(notDone()) {  
    prepare();  
    iterate();  
}  
reduce();
```



Making  
IT  
happen

Computing Science

# Mesh Pattern - CO<sub>2</sub>P<sub>3</sub>S

---

- Main loop:

```
initialize();  
while(notDone()) {  
    prepare();  
    iterate();  
}  
reduce();
```



# Mesh Pattern - CO<sub>2</sub>P<sub>3</sub>S

---

- Main loop:

```
initialize();  
while(notDone()) {  
    prepare();  
    iterate();  
}  
reduce();
```



# Mesh Pattern - CO<sub>2</sub>P<sub>3</sub>S

---

- Main loop:

```
initialize();  
while(notDone()) {  
    prepare();  
    iterate();  
}  
reduce();
```



Making  
IT  
happen

Computing Science

# Mesh Pattern - CO<sub>2</sub>P<sub>3</sub>S

---

- Main loop:

```
initialize();  
while(notDone()) {  
    prepare();  
    synchronize();  
    iterate();  
}  
reduce();
```



# Mesh Pattern - $MAP_3S$

---

- Same hooks, parameters as  $CO_2P_3S$
- C structures instead of user-defined classes
- explicit communication for structures
- macros instead of hook methods
  - explicit inlining



# Search-Tree Pattern

---

- Problems involving tree traversal
  - Optimization and heuristic search
  - Alpha-Beta search
- Task parallel problems

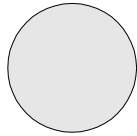


Making  
*IT*  
happen

Computing Science

# Search-Tree Pattern

---

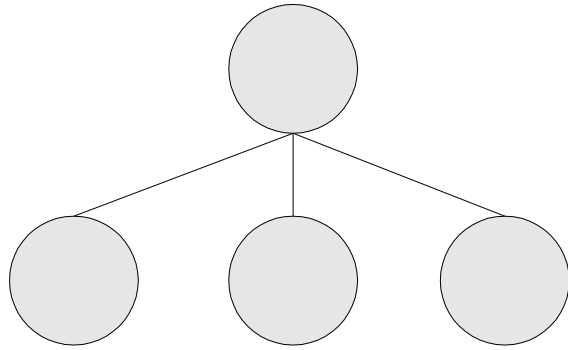


- Start from root



# Search-Tree Pattern

---

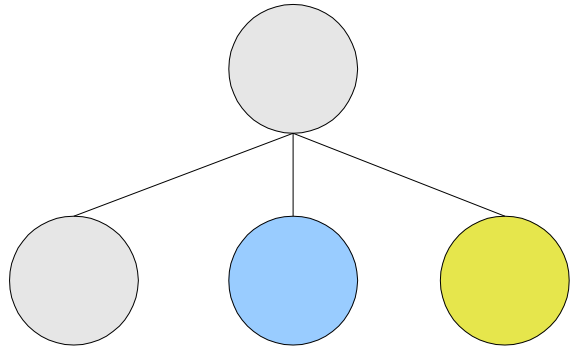


- Start from root
- Generate children



# Search-Tree Pattern

---



- Start from root
- Generate children
- Distribute children to multiple processes and compute sequentially



# Search-Tree - $CO_2P_3S$

---

- $CO_2P_3S$ 
  - tasks on a shared work queue
    - tasks represent subtrees to be searched
    - processes pull tasks off of shared queue as necessary



Making  
*IT*  
happen

Computing Science

# Search-Tree - $CO_2P_3S$

---

- parameters:
  - traversal technique (BFS vs. DFS)
  - early termination
- Main Hooks:
  - parallel
  - divide
  - conquer



# Search-Tree - CO<sub>2</sub>P<sub>3</sub>S

---

```
if (parallel ())  
    divide ();  
else  
    conquer ();
```



# Search-Tree - MAP<sub>3</sub>S

---

- divide and conquer methodology
- same parameters and hooks as CO<sub>2</sub>P<sub>3</sub>S
- more specific pattern for alpha-beta search
- dynamic load balancing



Making  
IT  
happen

Computing Science

# Search-Tree - MAP<sub>3</sub>S

---

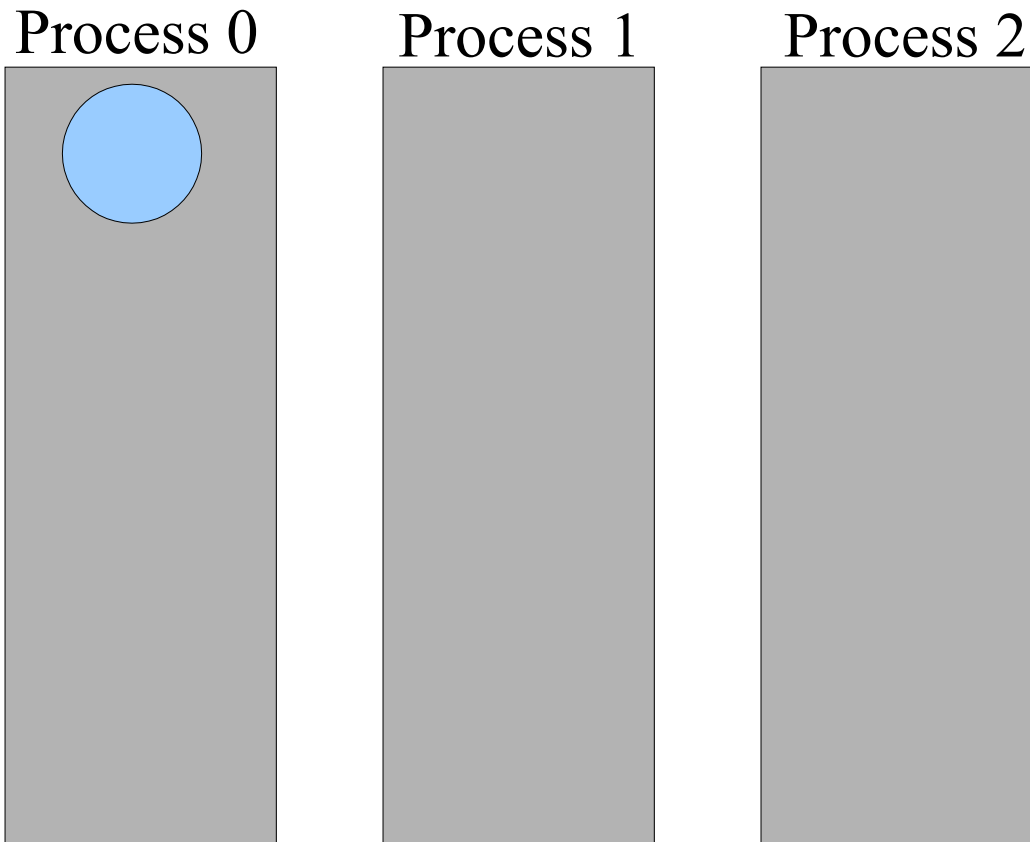
- Dynamic Load Balancing
  - multiple queues
  - load balancing by work-stealing



# Search-Tree - MAP<sub>3</sub>S

---

- Work stealing

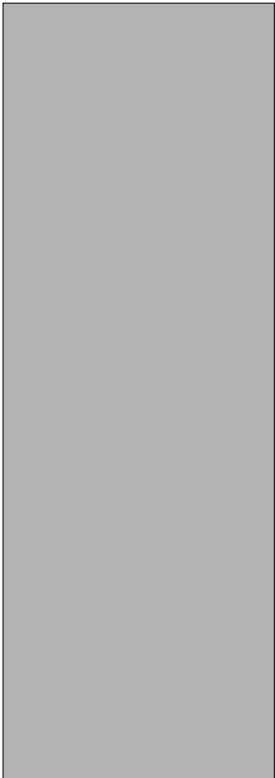


# Search-Tree - MAP<sub>3</sub>S

---

- Work Stealing

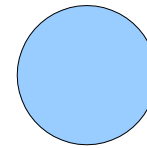
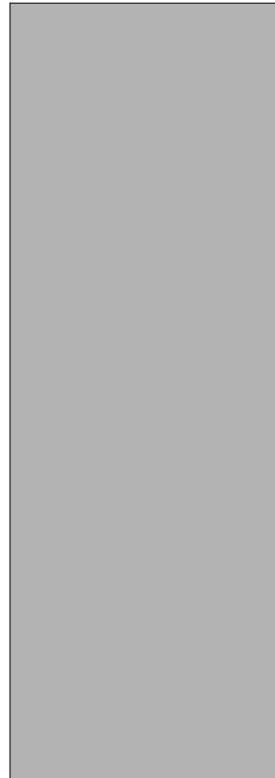
Process 0



Process 1



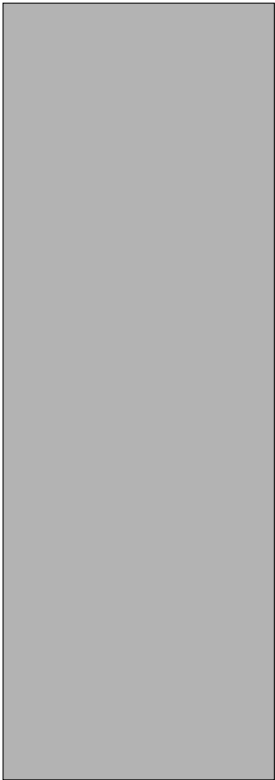
Process 2



# Search-Tree - MAP<sub>3</sub>S

- Work Stealing

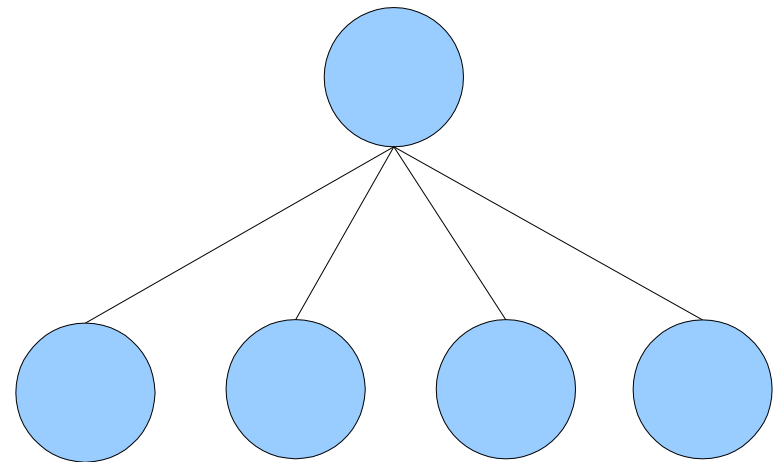
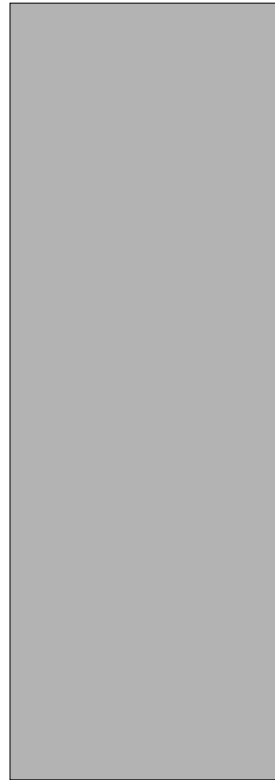
Process 0



Process 1



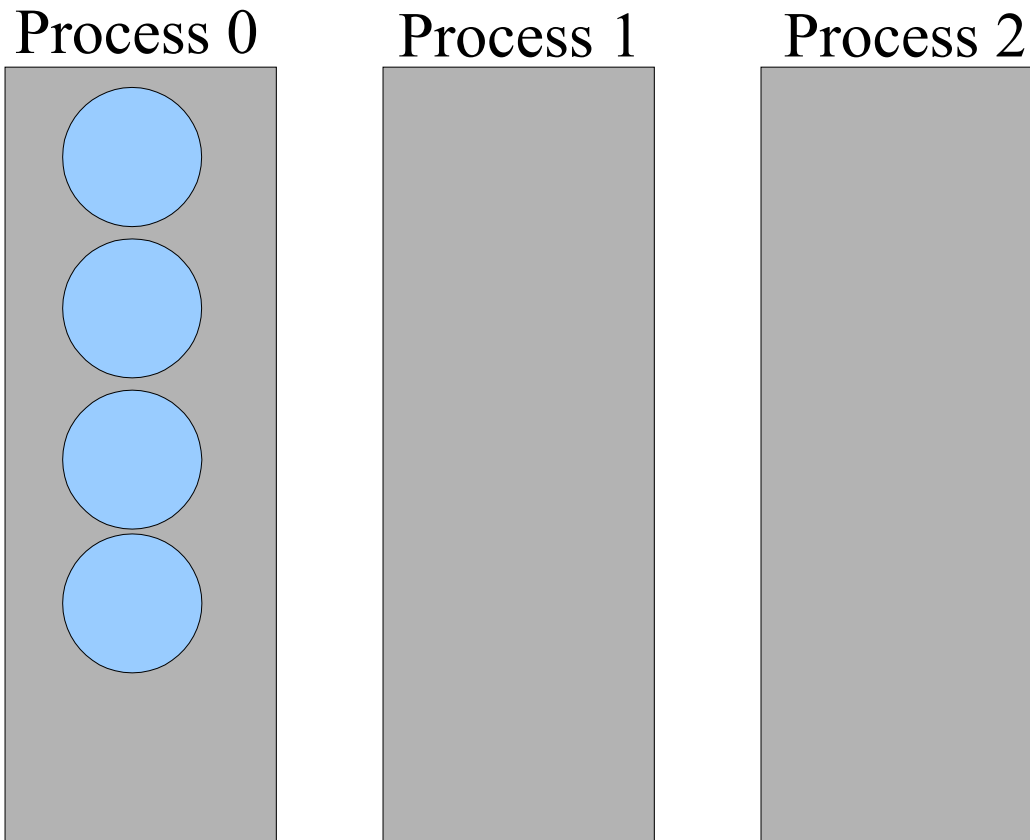
Process 2



# Search-Tree - MAP<sub>3</sub>S

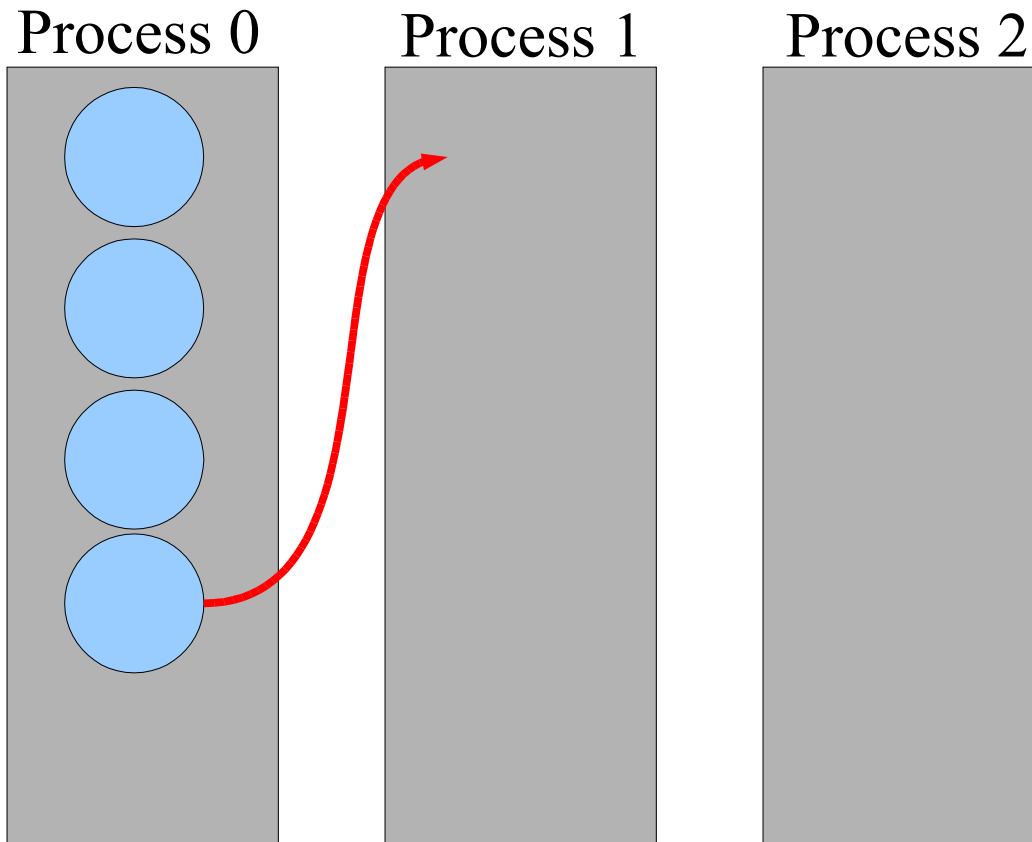
---

- Work Stealing



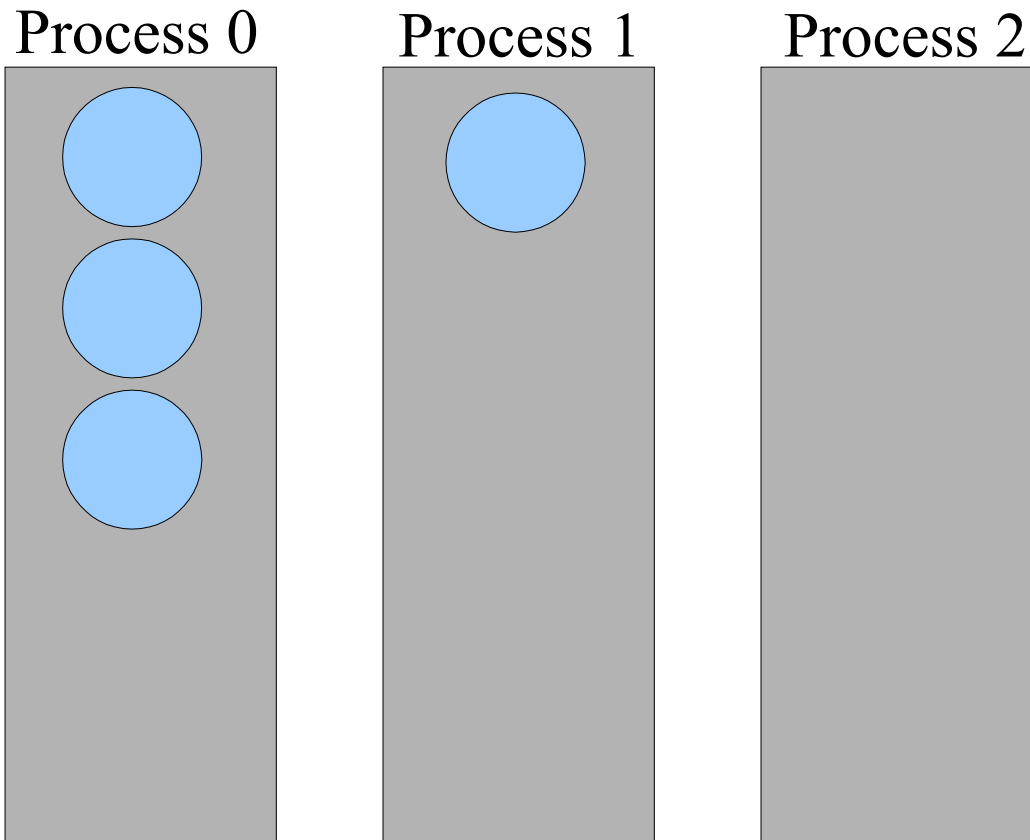
# Search-Tree - MAP<sub>3</sub>S

- Work Stealing



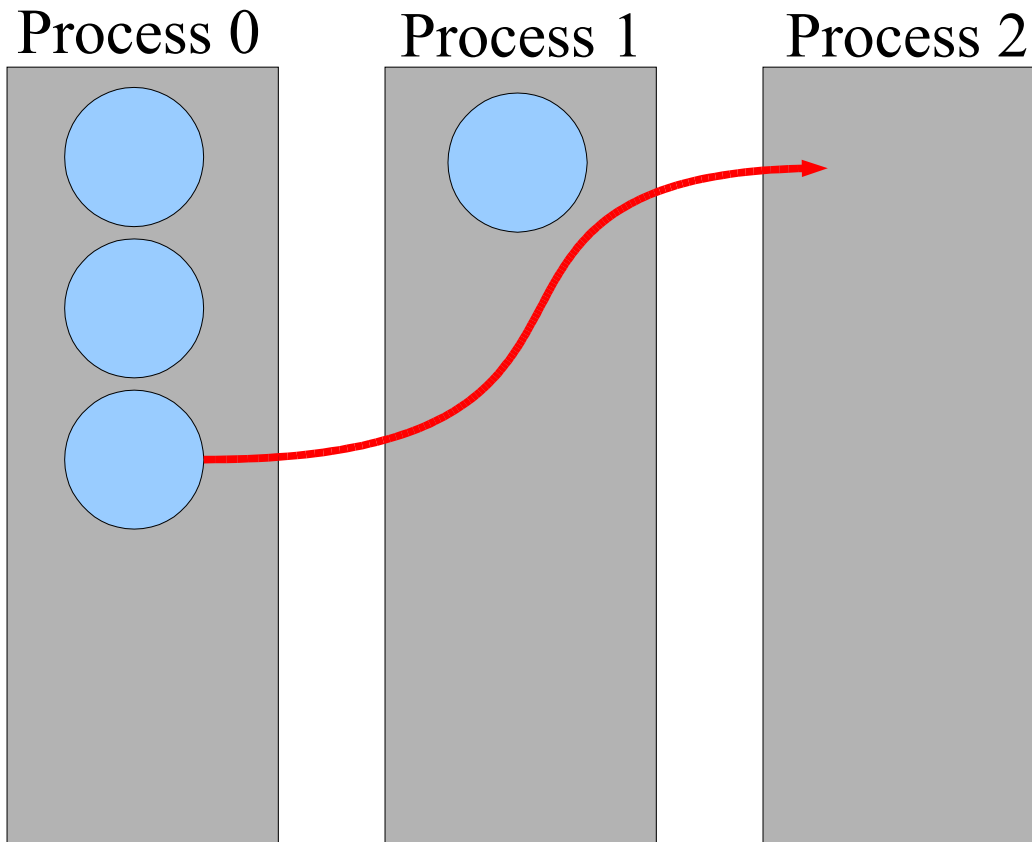
# Search-Tree - MAP<sub>3</sub>S

- Work Stealing



# Search-Tree - MAP<sub>3</sub>S

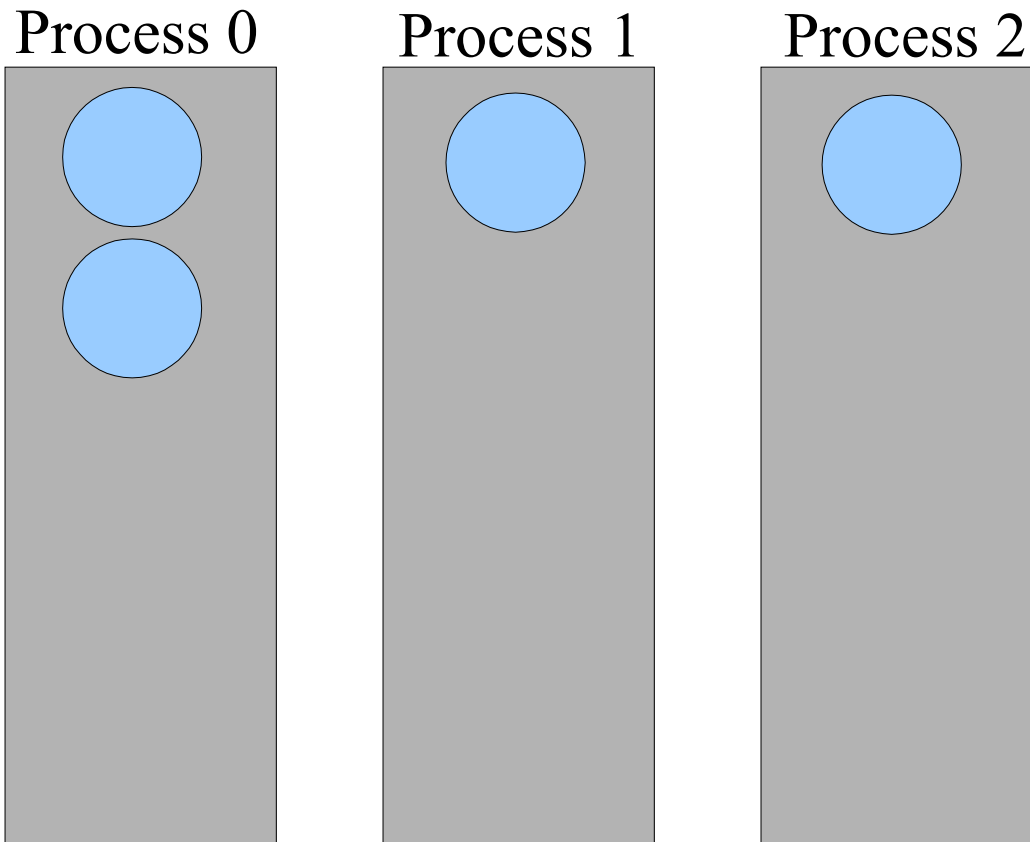
- Work Stealing



# Search-Tree - MAP<sub>3</sub>S

---

- Work Stealing



# Search-Tree - MAP<sub>3</sub>S

---

- Dynamic Load Balancing
  - comprises a task farm
  - useful for many task-parallel problems
  - adapted for use with the search-tree



# Search-Tree - MAP<sub>3</sub>S

- Computation Thread:

```
if (!workListEmpty()) {  
    getWork();  
    processTask();  
}  
else {  
    requestTask();  
    wait(request);  
}
```



# Search-Tree - MAP<sub>3</sub>S

- Computation Thread:

```
if (!workListEmpty()) {  
    getWork();  
    processTask();  
}  
else {  
    requestTask();  
    wait(request);  
}
```



# Search-Tree - MAP<sub>3</sub>S

- Computation Thread:

```
if (!workListEmpty()) {  
    getWork();  
    processTask();  
}  
else {  
    requestTask();  
    wait(request);  
}
```



# Search-Tree - MAP<sub>3</sub>S

- Computation Thread:

```
if (!workListEmpty()) {  
    getWork();  
    processTask();  
}  
else {  
    requestTask();  
    wait(request);  
}
```



# Search-Tree - MAP<sub>3</sub>S

---

- Message Processing Thread:

```
while (!taskFarmDone) {  
    MPI_Probe();  
    processMessage();  
}
```



# Search-Tree - MAP<sub>3</sub>S

---

- Message Processing Thread:

```
while (!taskFarmDone) {  
    MPI_Probe ();  
    processMessage ();  
}
```



# Search-Tree - MAP<sub>3</sub>S

---

- Message Processing Thread:

```
while (!taskFarmDone) {  
    MPI_Probe();  
    processMessage();  
}
```



# Search-Tree - MAP<sub>3</sub>S

---

- Message Processing Thread:

```
while (!taskFarmDone) {  
    MPI_Probe ();  
    processMessage ();  
}
```



# Search-Tree - MAP<sub>3</sub>S

---

- Message Processing Thread:

```
while (!taskFarmDone) {  
    MPI_Probe();  
    processMessage();  
    // receive requests  
}
```



# Search-Tree - MAP<sub>3</sub>S

---

- Message Processing Thread:

```
while (!taskFarmDone) {  
    MPI_Probe();  
    processMessage();  
    // receive requested tasks  
}
```



# Search-Tree - MAP<sub>3</sub>S

---

- Message Processing Thread:

```
while (!taskFarmDone) {  
    MPI_Probe();  
    processMessage();  
    // termination cycle  
}
```



# Results

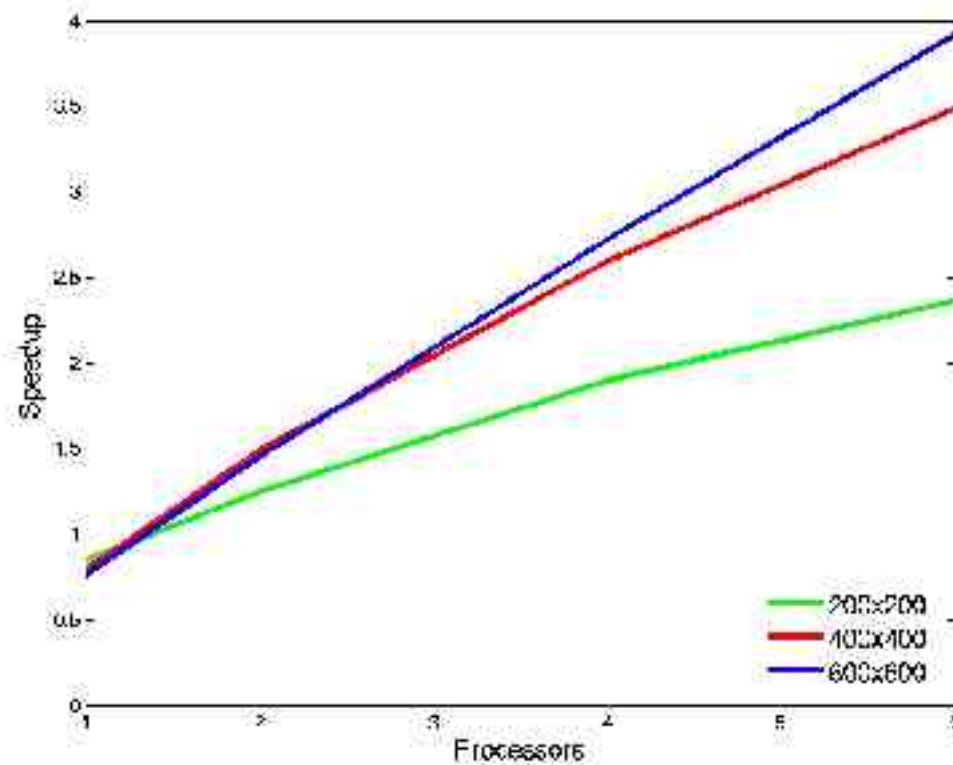
---

- Usability
  - For the implemented patterns,  $MAP_3S$  recreates the hooks of  $CO_2P_3S$
  - Pattern-user needs write only sequential, computational code
  - Generated communication framework handles the rest



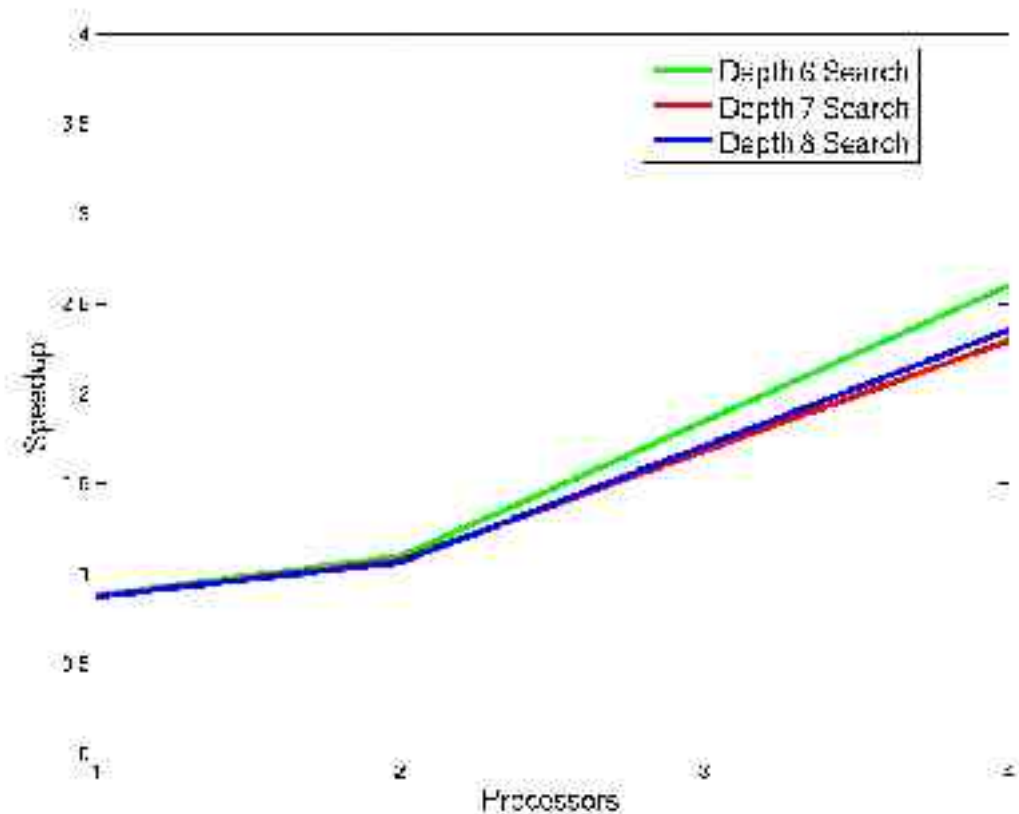
# Results

- Speedup – Mesh Pattern:



# Results

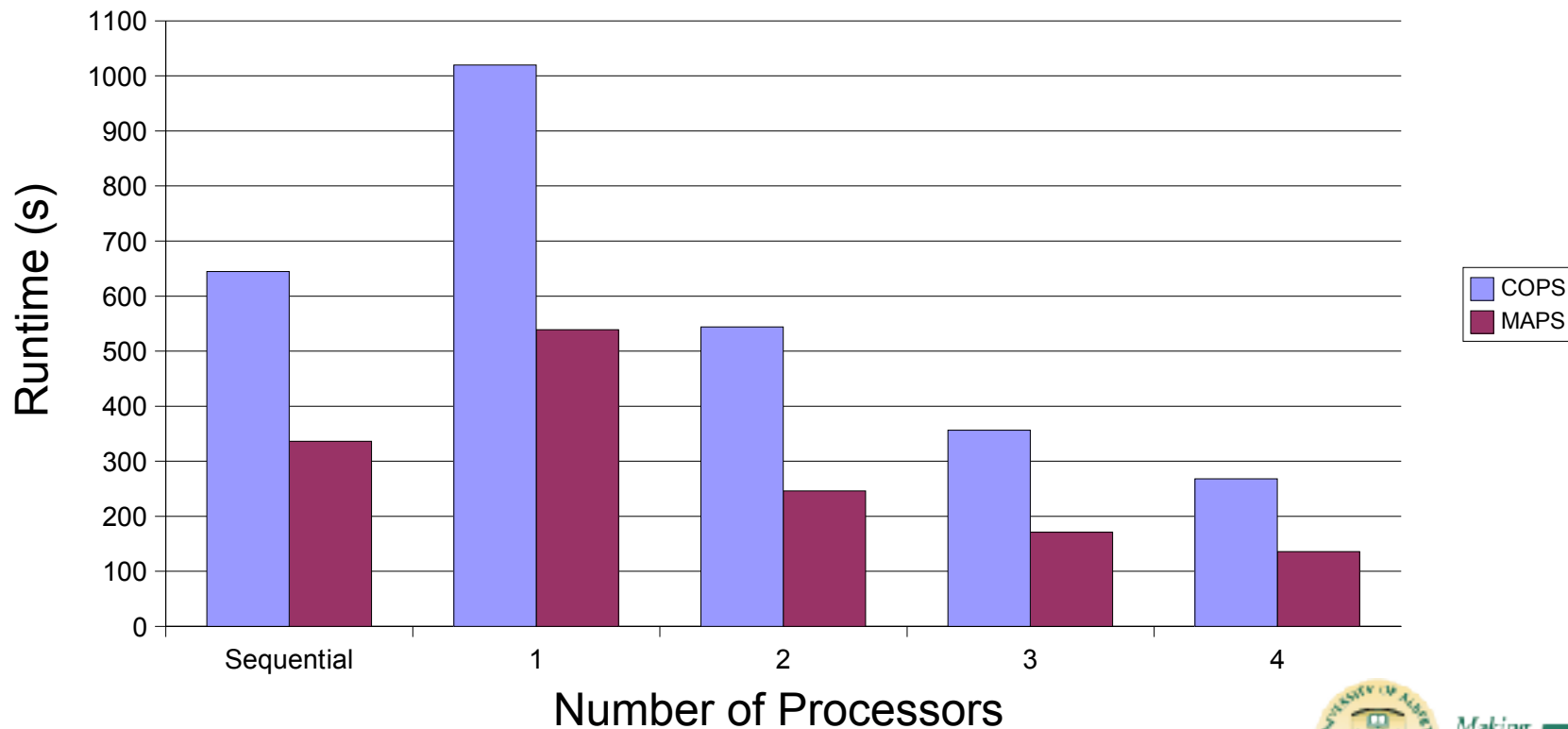
- Speedup – Search-Tree Pattern:



# Results

- Comparison –  $CO_2P_3S$  vs.  $MAP_3S$

Conway's Game of Life - Shared Memory



# Immediate-Future Work

---

- Current Patterns
  - test more problems
- New Patterns
  - continue Wavefront
  - develop Pipeline



# Future Work

---

- Current Patterns
  - test with more users
  - performance tuning
- More Patterns
- Pattern-Framework Generator
- Standardize pattern hooks
  - allow for completely different frameworks



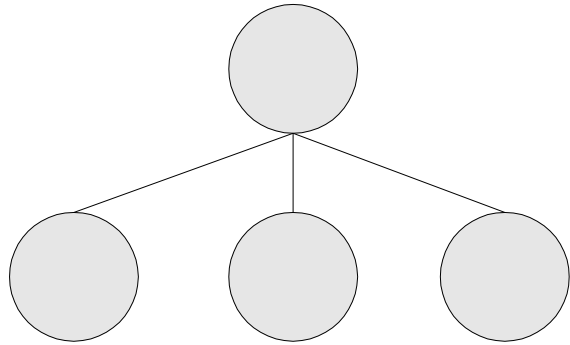
# Questions?

---



# Search-Tree - $MAP_3S$

---

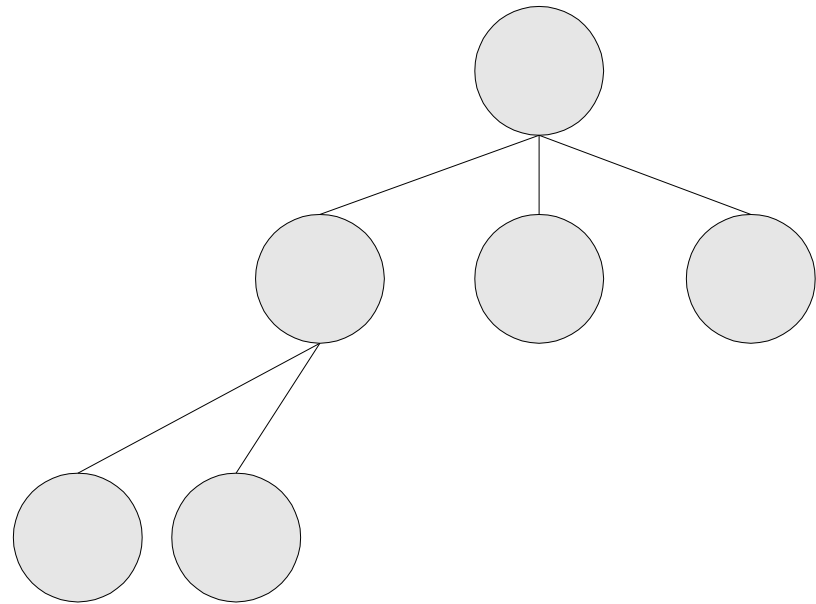


- Start from root
- Generate to given depth



# Search-Tree - $MAP_3S$

---



- Start from root
- Generate to given depth



# Search-Tree - $MAP_3S$

---

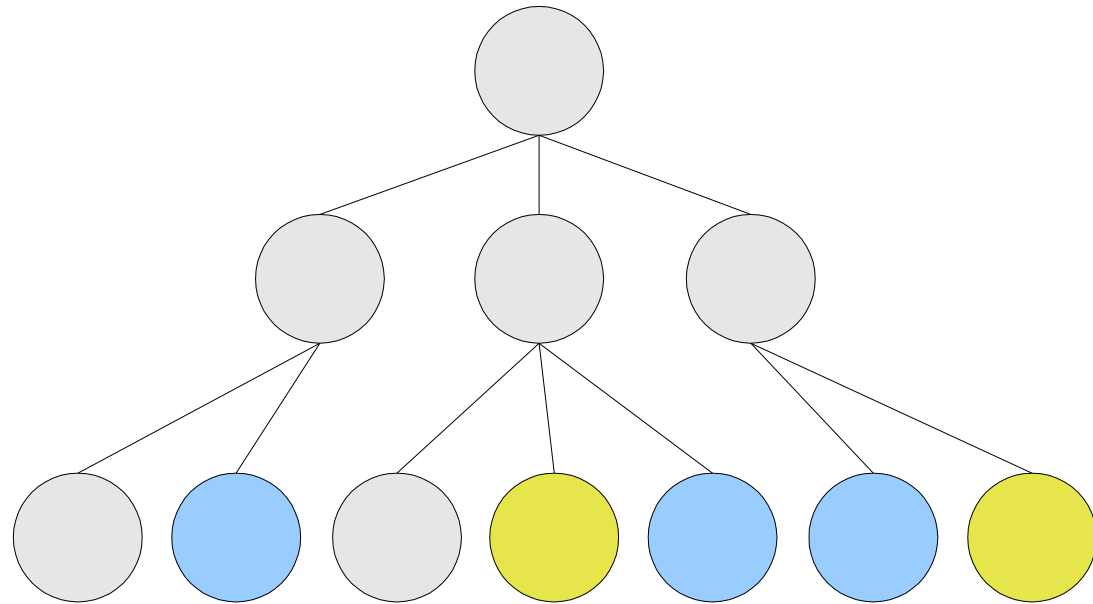


- Start from root
- Generate nodes to given depth
- Processes steal leaf nodes from generating process



# Search-Tree - $\text{MAP}_3 S$

---



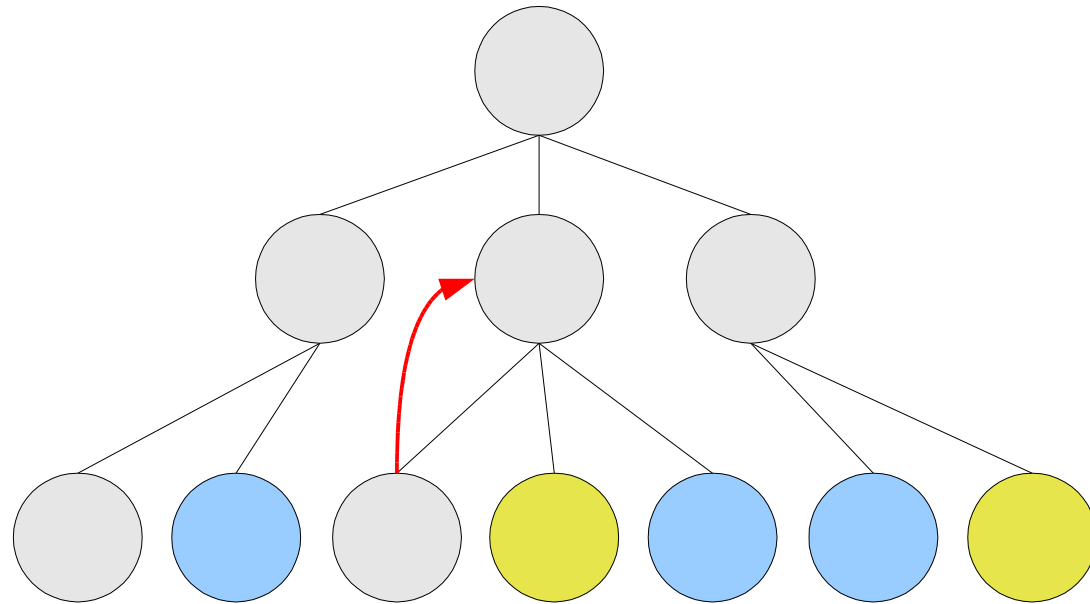
- Leaf nodes represent subtrees to be processed sequentially



Making  
IT  
happen

Computing Science

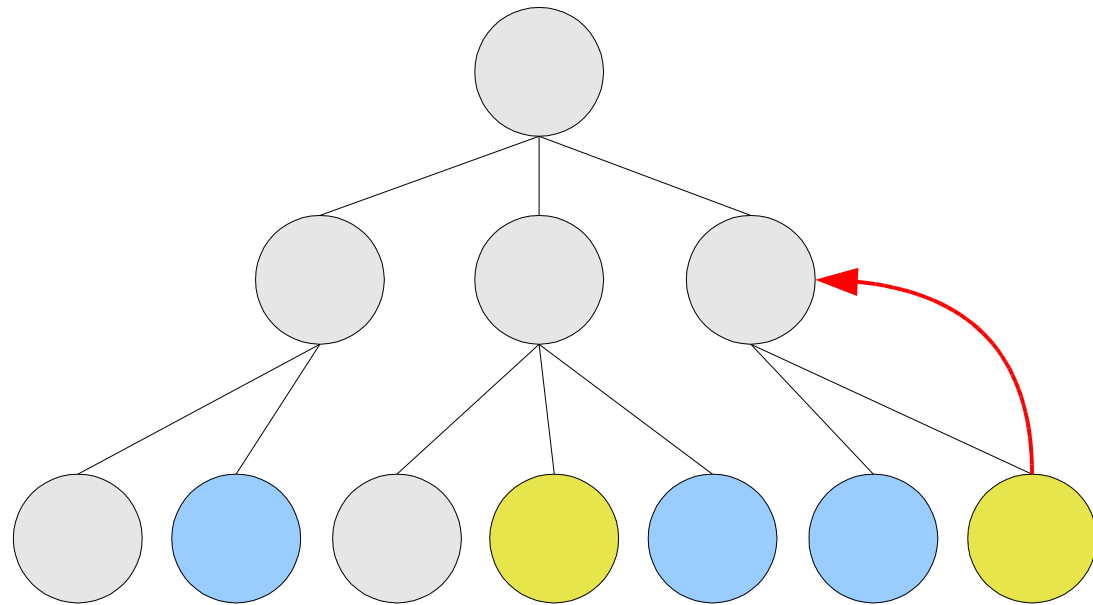
# Search-Tree - $MAP_3S$



- Completed leaf nodes return computational result to parent



# Search-Tree - MAP<sub>3</sub>S



- Completed leaf nodes return computational result to parent
- Updating parent may require message

