Parallel(la) Programming

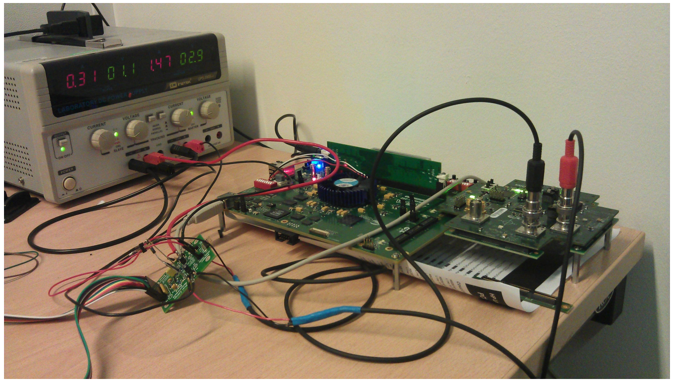
Concurrency, Deadlock and Interconnect

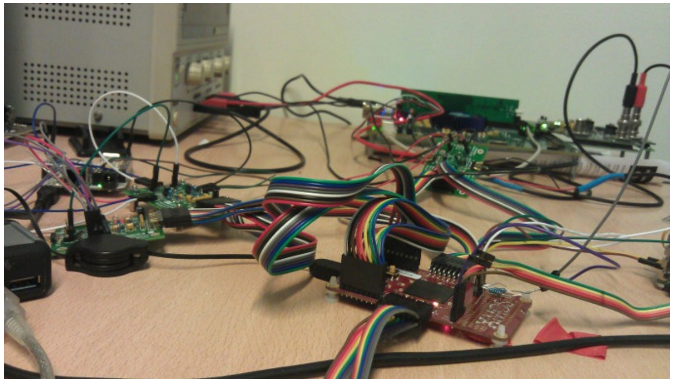
James Pallister
Research Engineer, Embecosm
PhD Student, University of Bristol

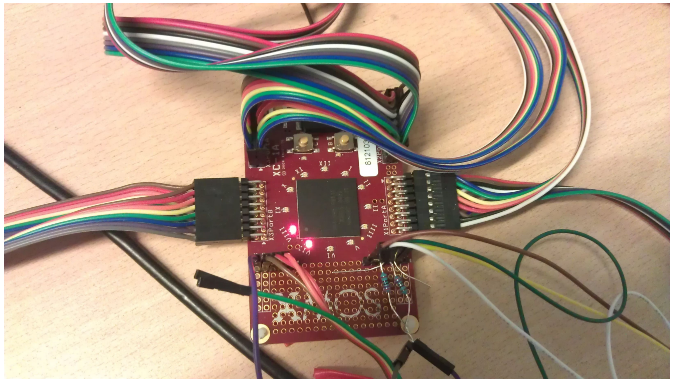
EMBECOSM®

About me

- PhD studying *software-based* energy efficiency.
  - Currently looking at compiler technology
- Last summer:







Outline

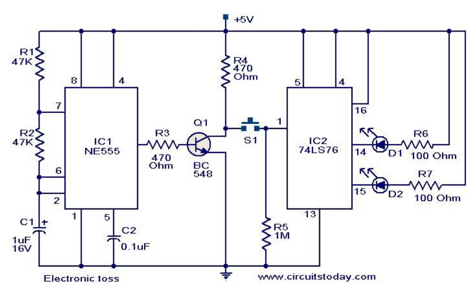
Parallelism

Deadlock & race conditions

Epiphany's interconnect

Why Parallel?

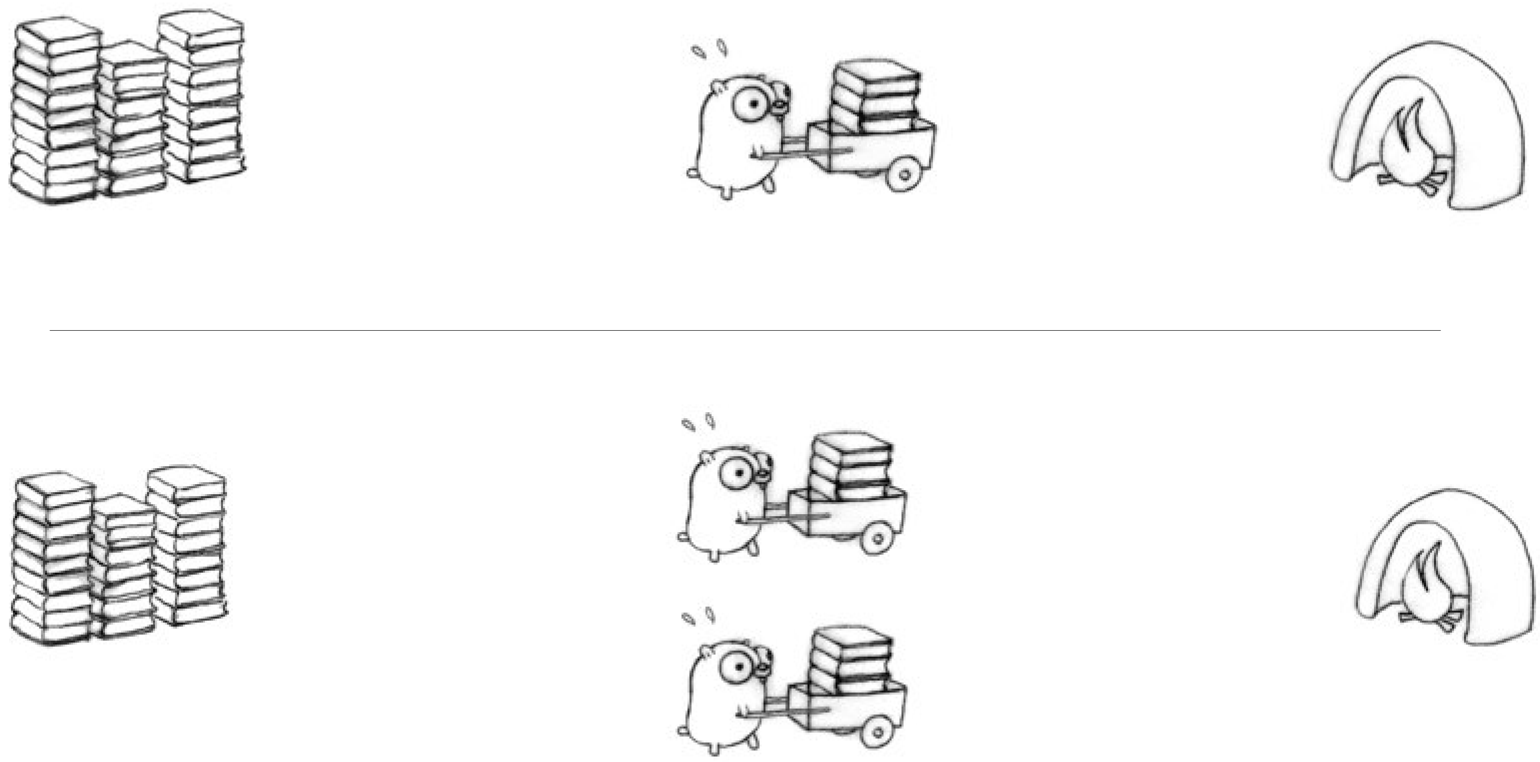
The world is parallel.



But when programming, we've been taught to think sequentially.

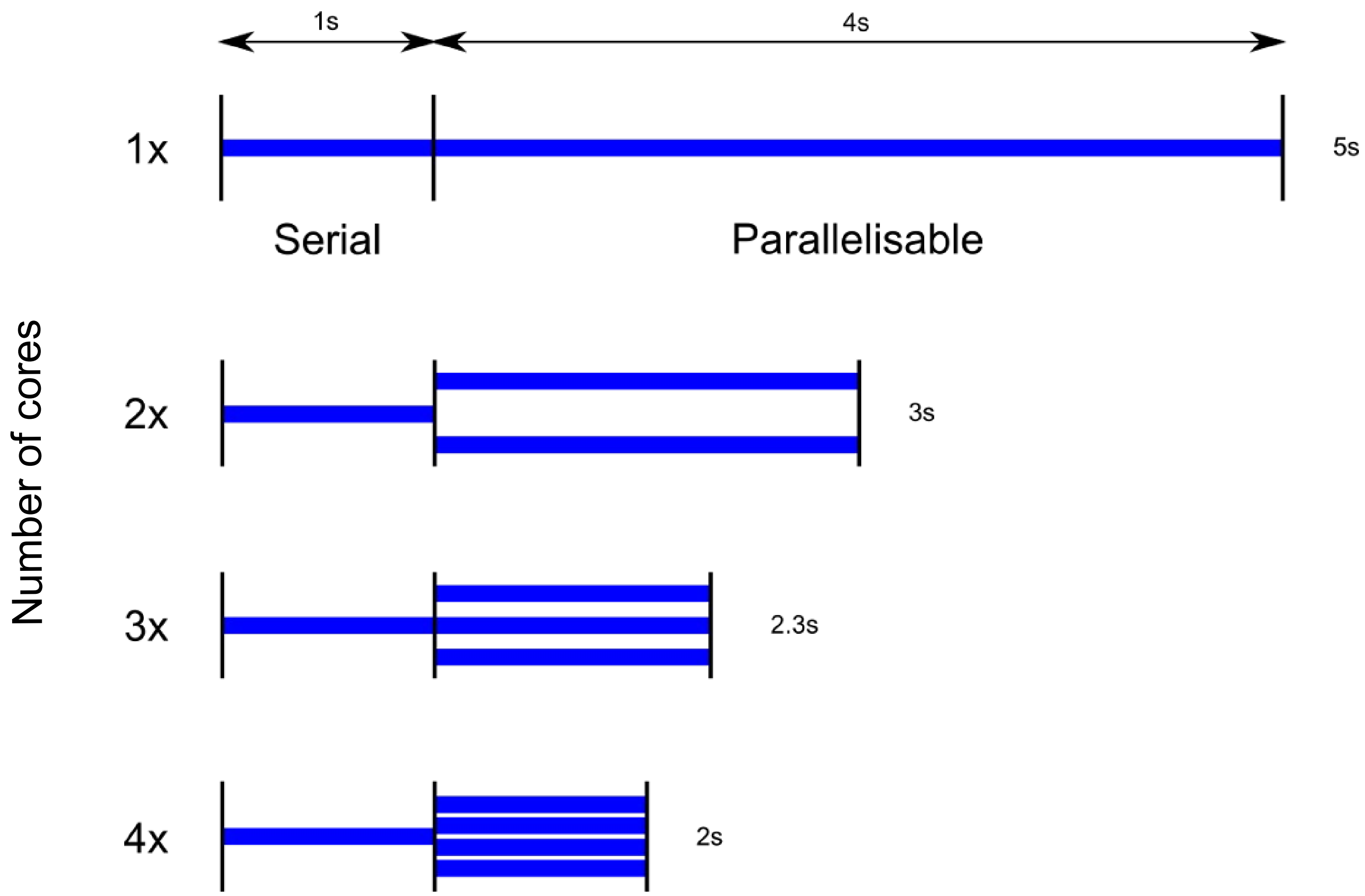
What is Parallelism?

- Multiple things happening at the same time to progress towards a common goal.

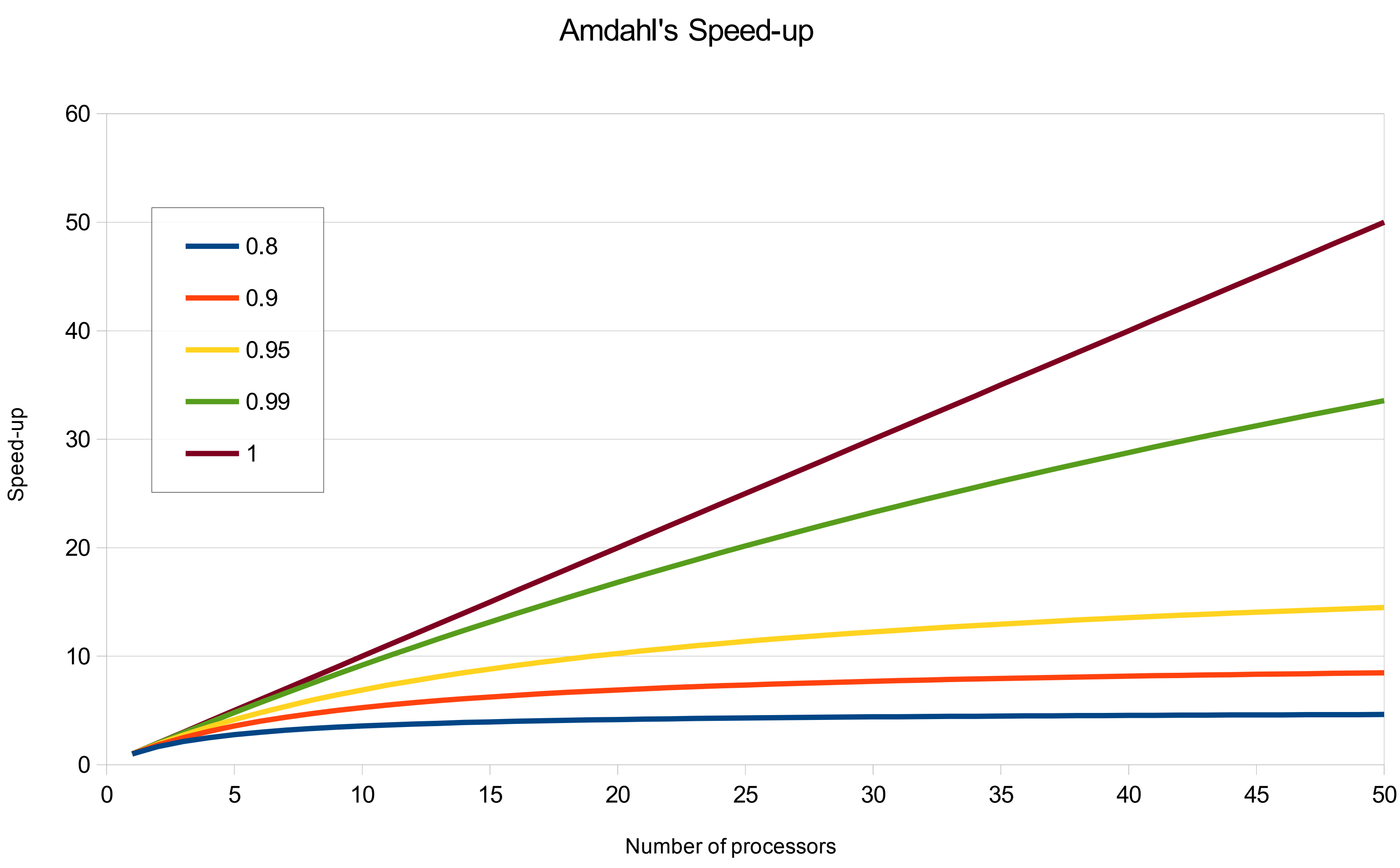


EMBECOSM®

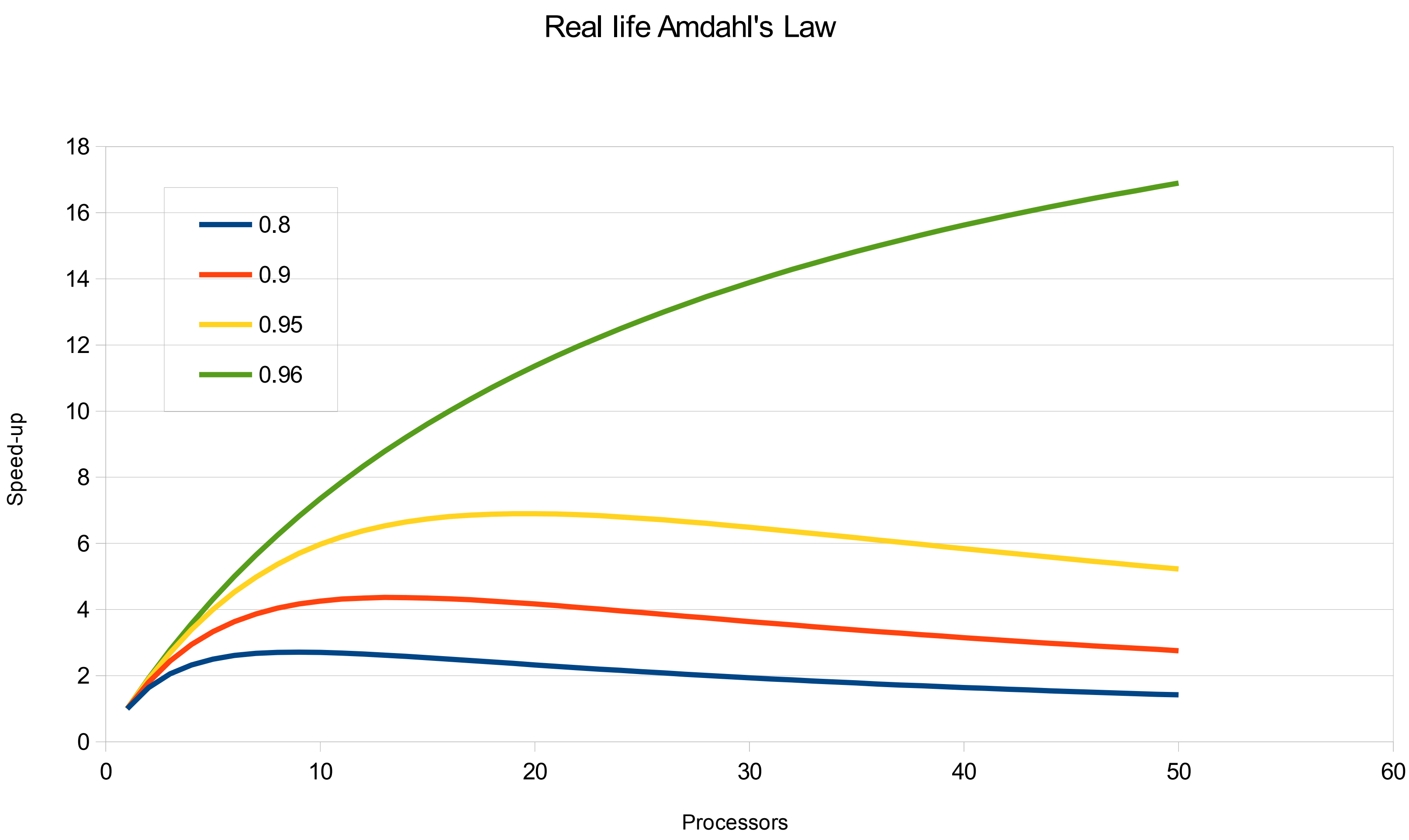
Amdahl's Law


1s
4s
1x
5s
Serial
Parallelisable
Number of cores
2x
3s
3x
2.3s
4x
2s

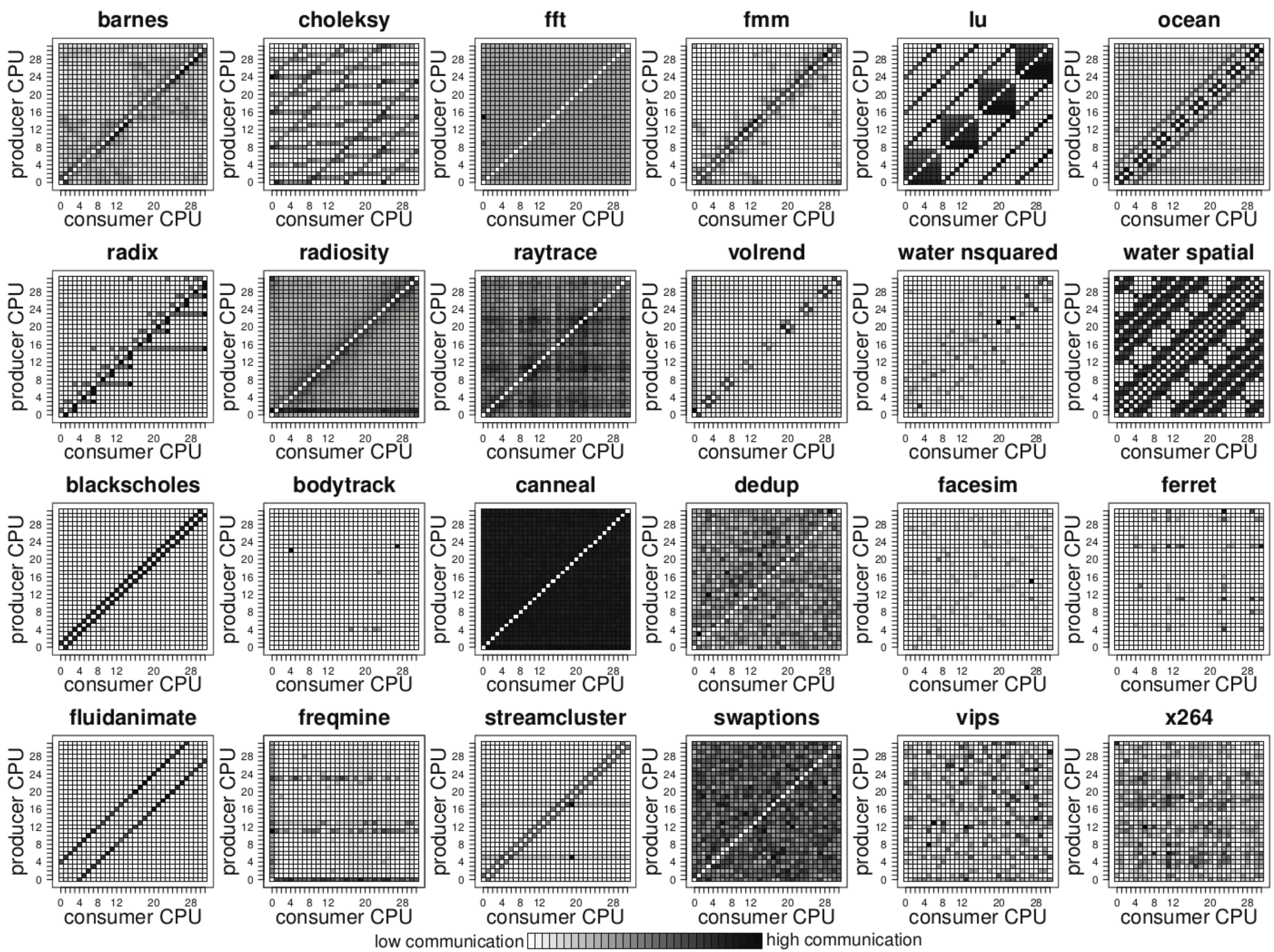
Amdahl's Law


Amdahl's Speed-up
Speed-up
Number of processors
0
10
20
30
40
50
60
0
5
10
15
20
25
30
35
40
45
50
0.8
0.9
0.95
0.99
1

Real life Amdahl's Law

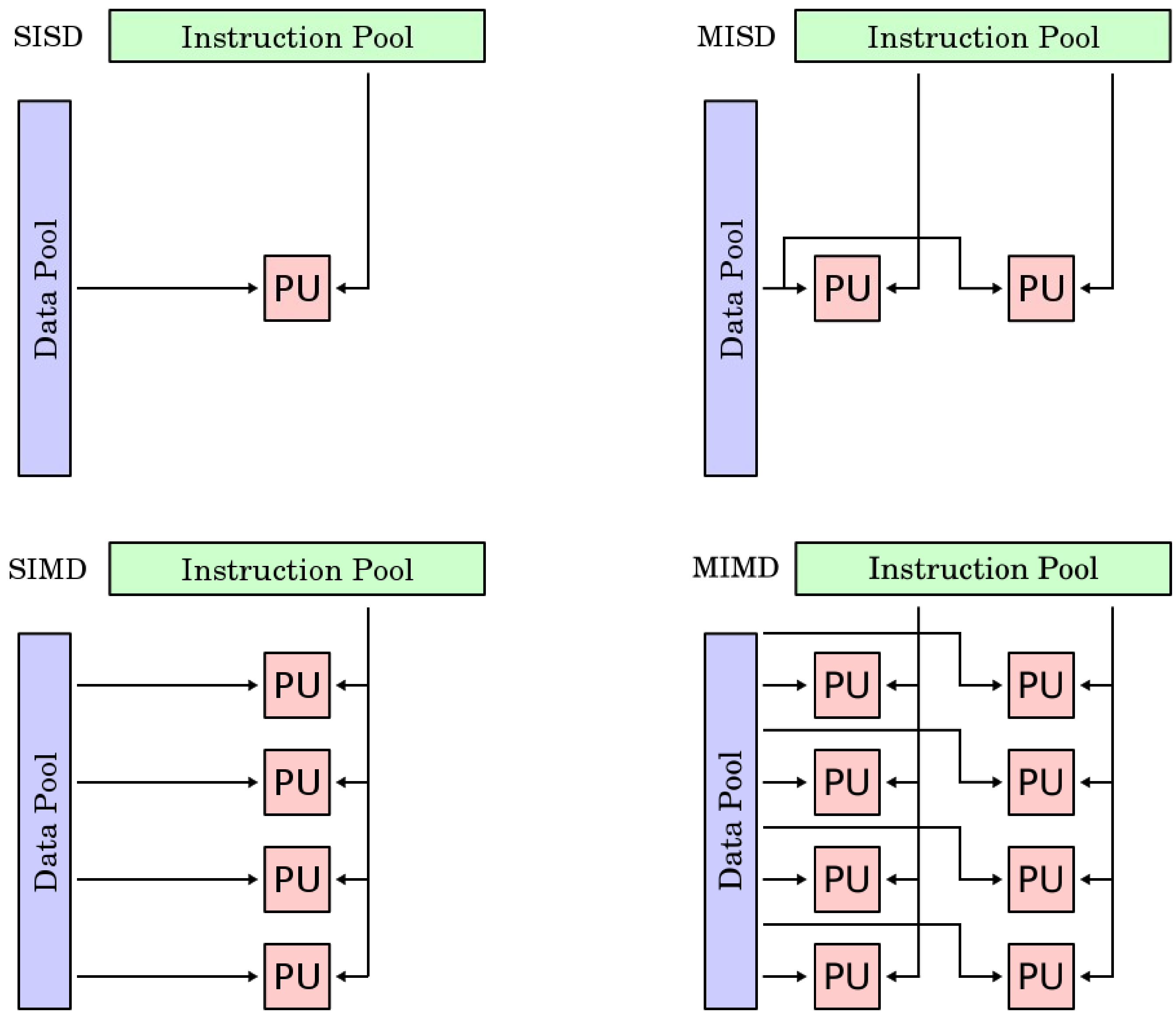


Communication Patterns

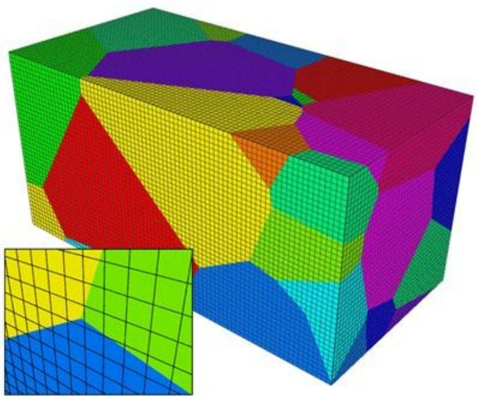


Fensch, C., Barrow-Williams, N., Mullins, R. D., & Moore, S. W. (2012). Designing a Physical Locality Aware Coherence Protocol for Chip-Multiprocessors. IEEE Transactions on Computers.

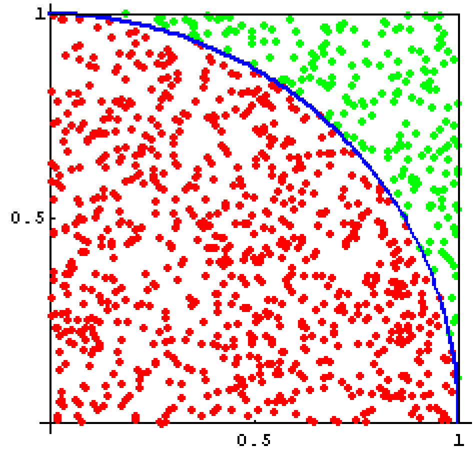
Flynn's Taxonomy



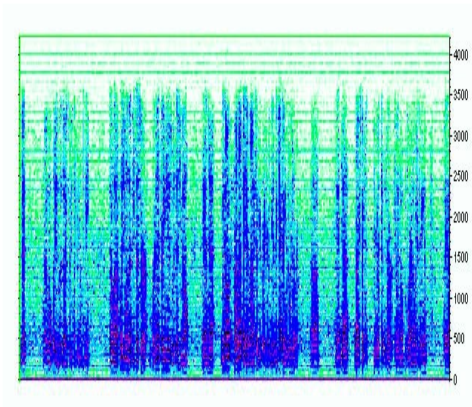
The 7 dwarves of parallel computation



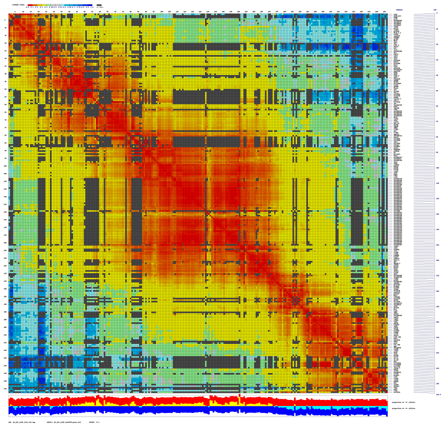
Structured Grid



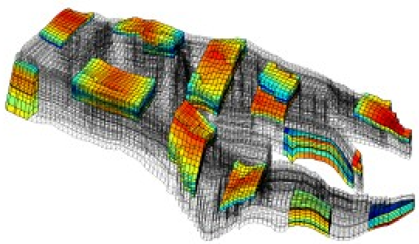
Monte Carlo



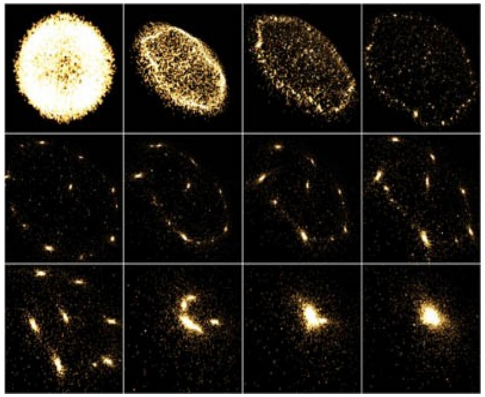
Spectral



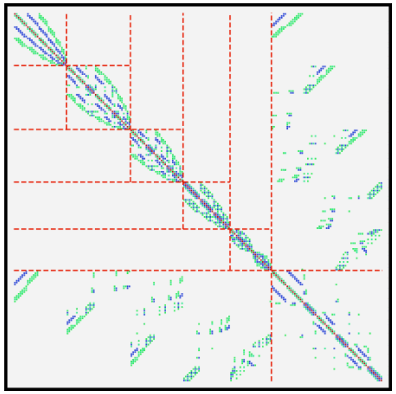
Dense Linear Algebra



Unstructured Grid



N-Body



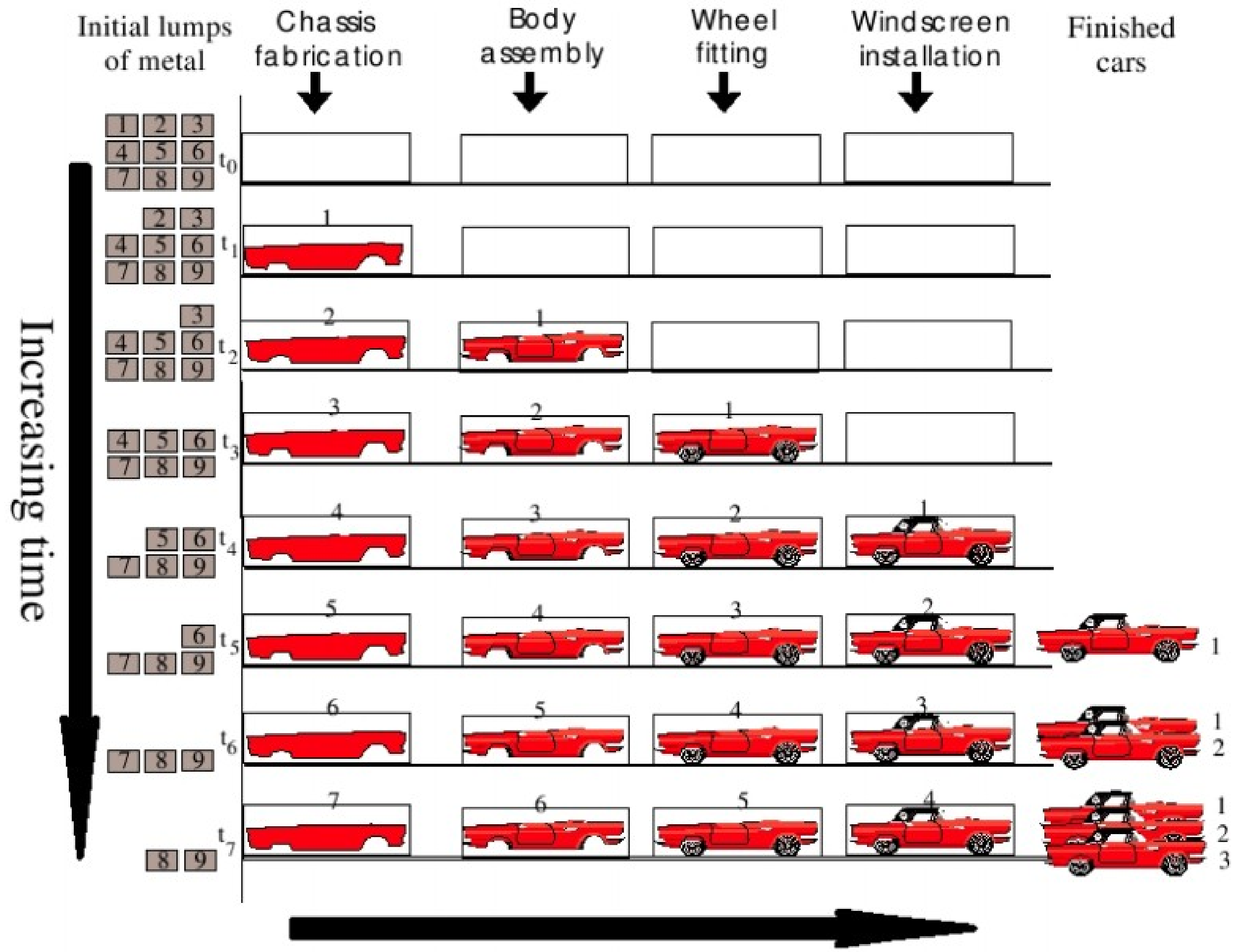
Sparse Linear Algebra

EMBECOSM®

Parallelism - Paradigms

- Pipelines
- Task farms
  - Server - client
- Geometric
  - Matrix multiply
  - Structured grid

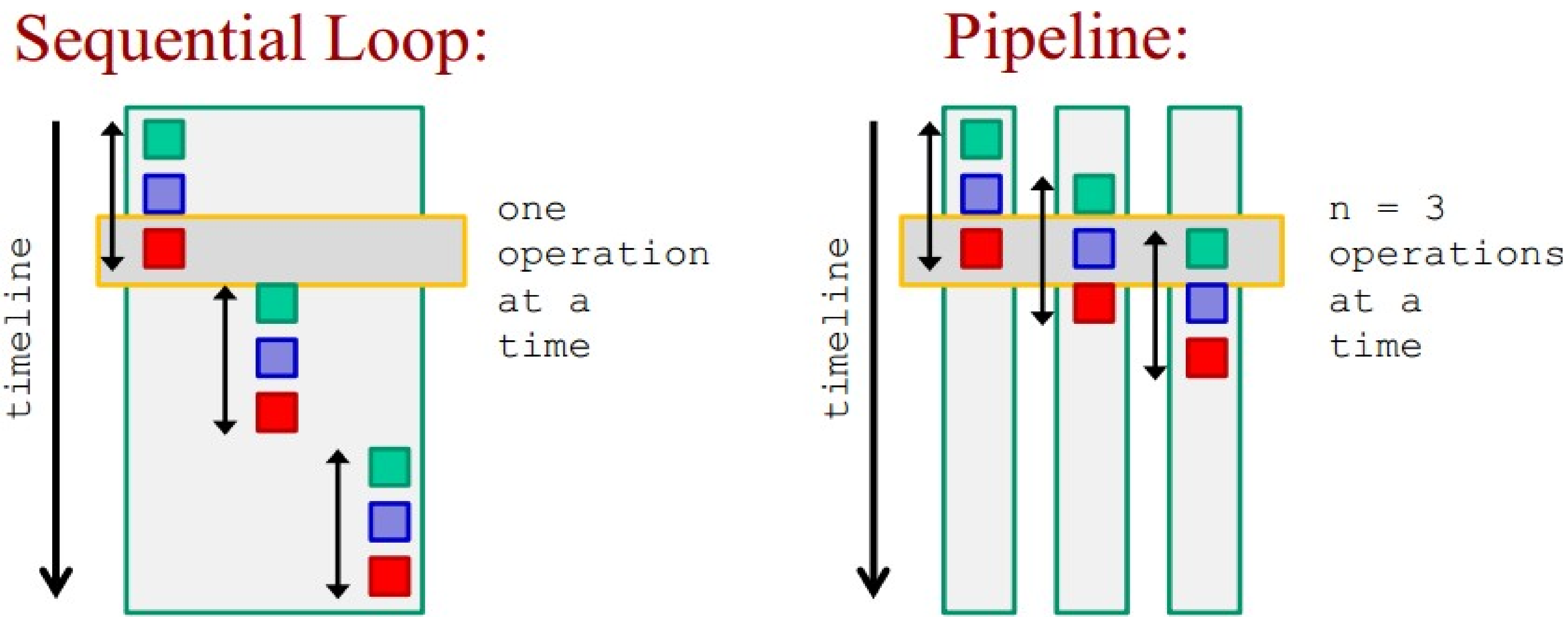
Pipelines



Motion of cars through pipeline

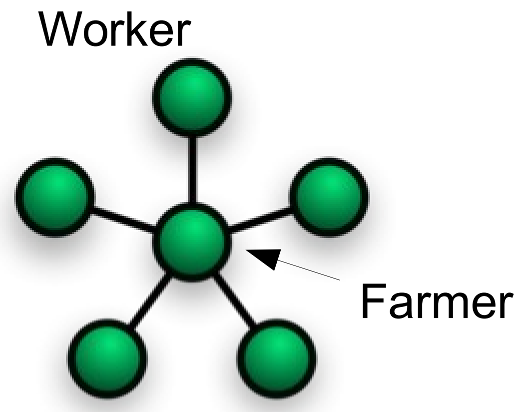
from "Alan Chalmers, Practical Parallel Processing, 1996"

Pipelining


Sequential Loop:
timeline
one
operation
at a
time
Pipeline:
timeline
n = 3
operations
at a
time

Task Farms

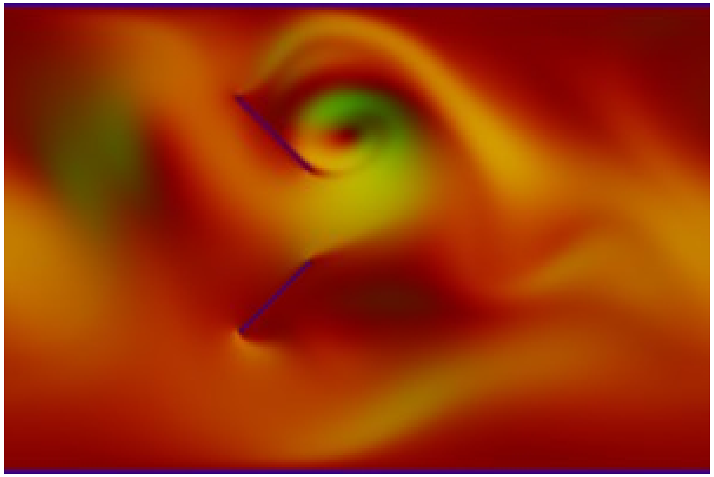
- The 'farmer' distributes work
- The workers do the work, and when finished ask for the next item.
- The farmer does a lot of communication – can become communication bound

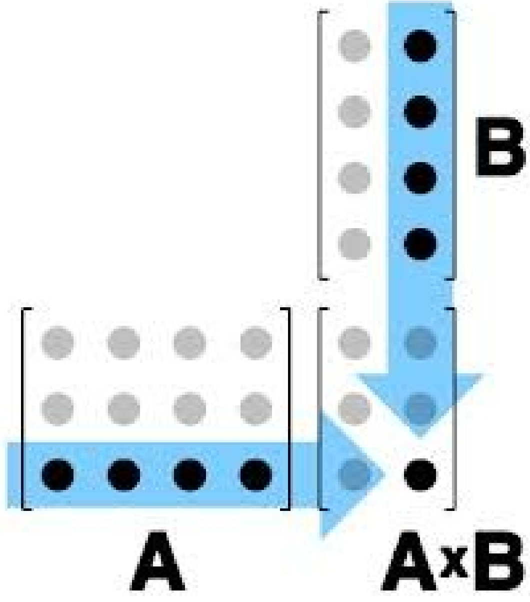


EMBECOSM®

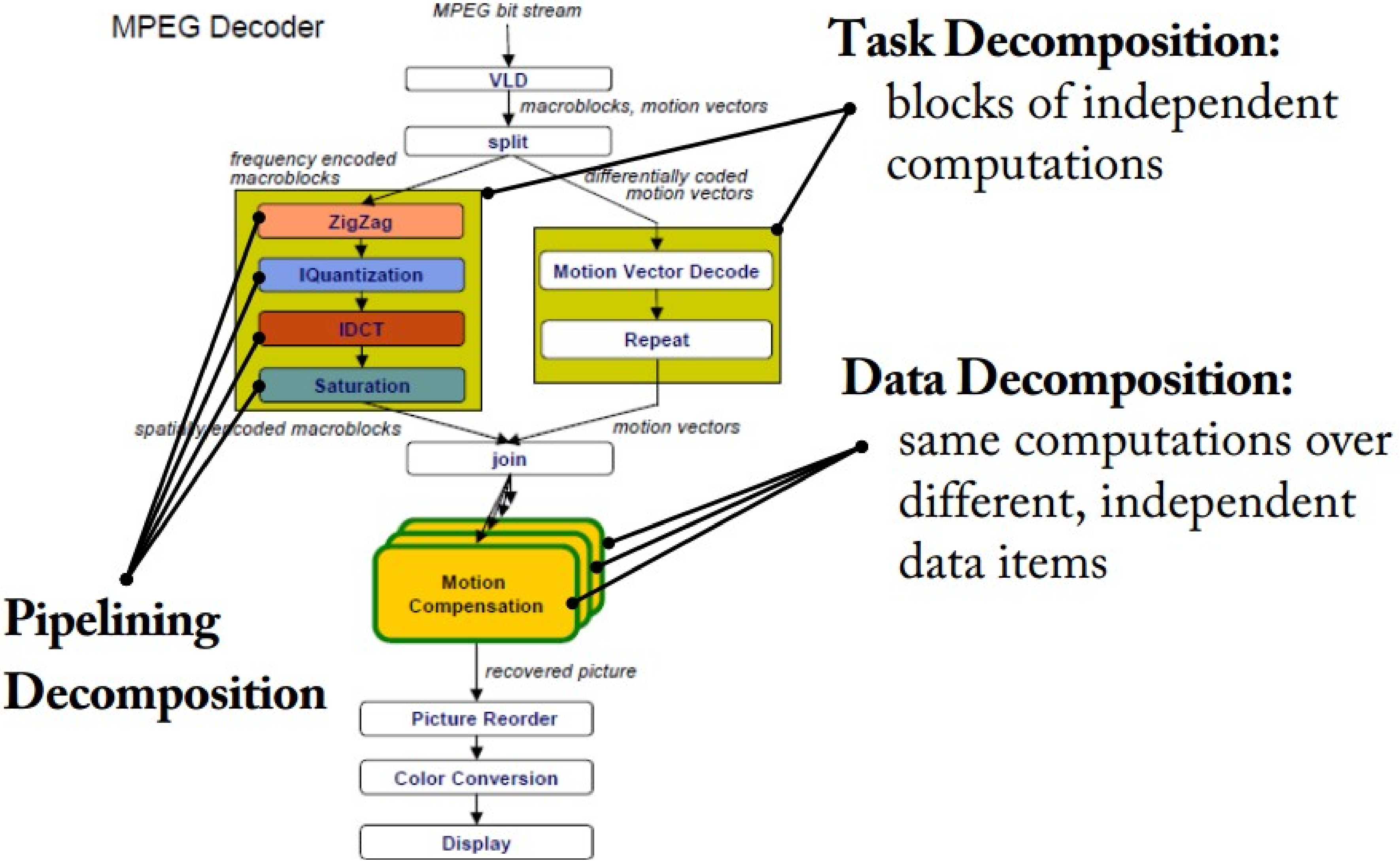
Geometric Parallelism

- Often based on the layout of the task
- Matrix multiply
  - Grid-based
- Physical simulation
  - Often grid-based



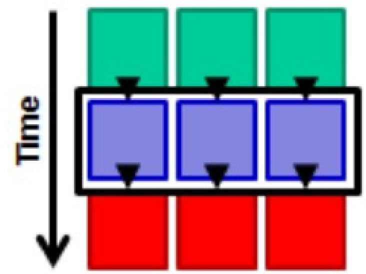


An Example

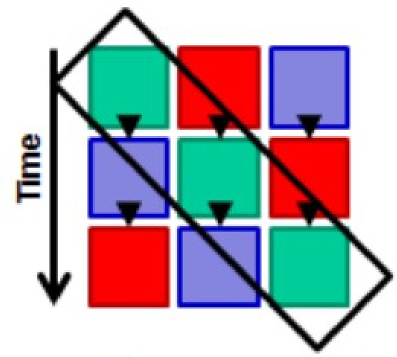


MPEG Decoder
MPEG bit stream
VLD
macroblocks, motion vectors
split
frequency encoded macroblocks
differentially coded motion vectors
ZigZag
IQuantization
IDCT
Saturation
Motion Vector Decode
Repeat
spatially encoded macroblocks
motion vectors
join
Motion Compensation
recovered picture
Picture Reorder
Color Conversion
Display
Task Decomposition:
blocks of independent computations
Data Decomposition:
same computations over different, independent data items
Pipelining Decomposition

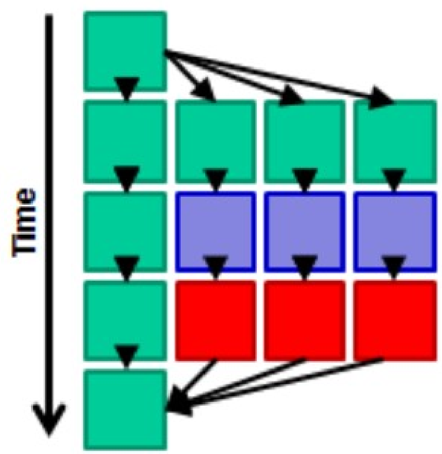
More Examples



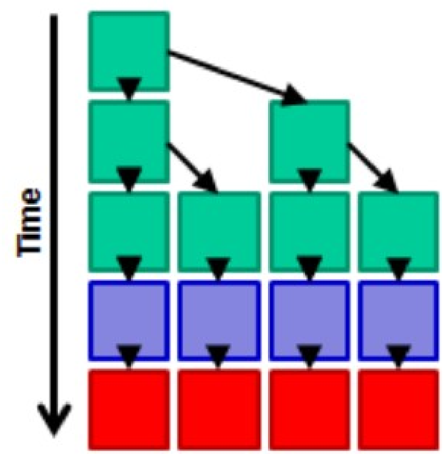
SIMD Parallelism



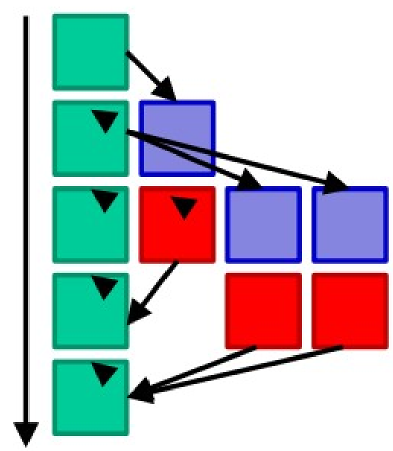
Pipelining (Chain Process)



Replication

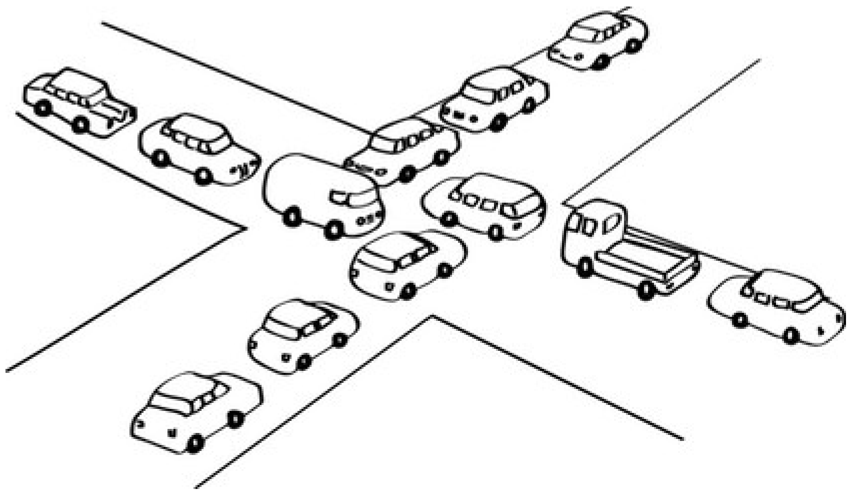


Hierarchical Fork



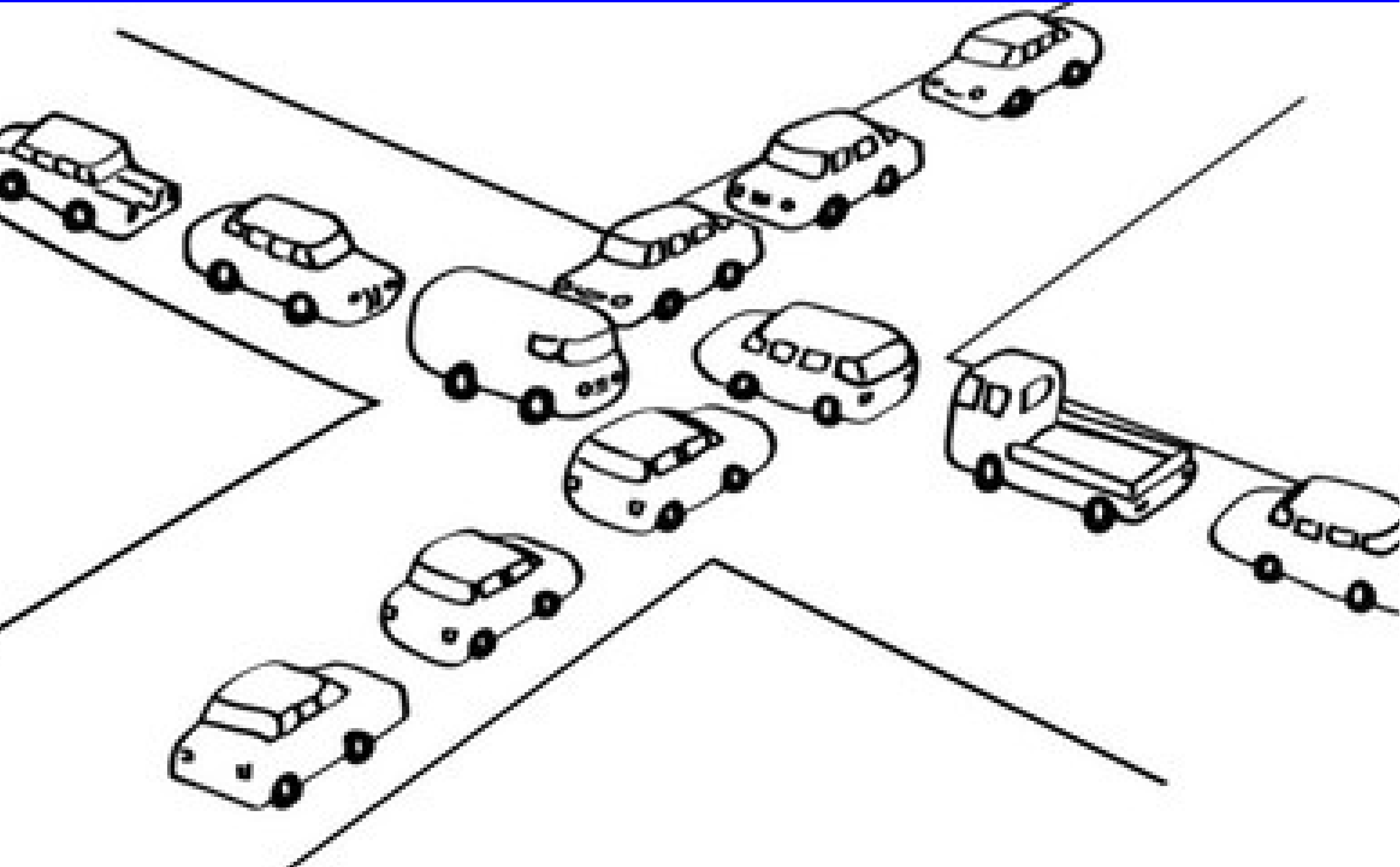
Farming in Star Topology

Deadlock and Race Conditions



http://csunplugged.org/routing-and-deadlock

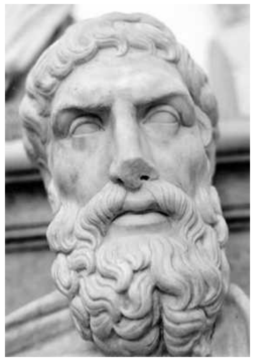
The Dining Philosophers



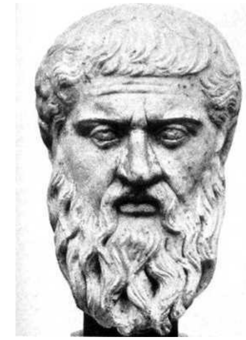
Conditions for deadlock

- Mutual exclusion
- Resource holding
- No pre-emption
- Circular waiting

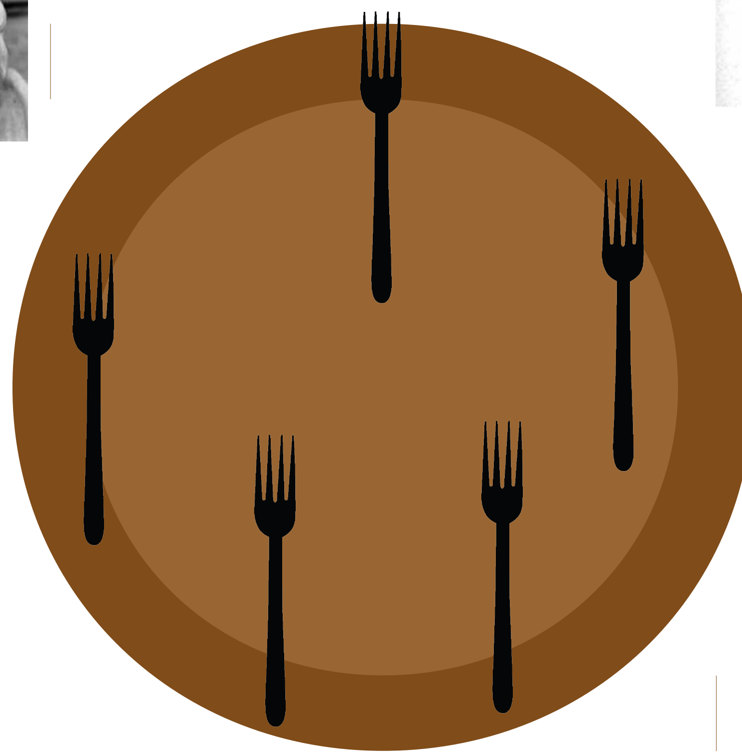
The Dining Philosophers



Epicurus

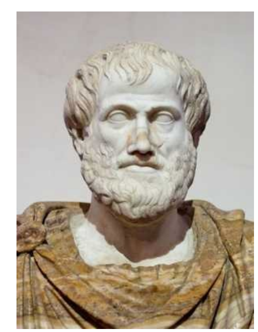


Plato



Paul of
Tarsus

Descartes



Aristotle

Race Conditions

Race Conditions

Thread 1

```
printf("hello ");
printf("world ");
```

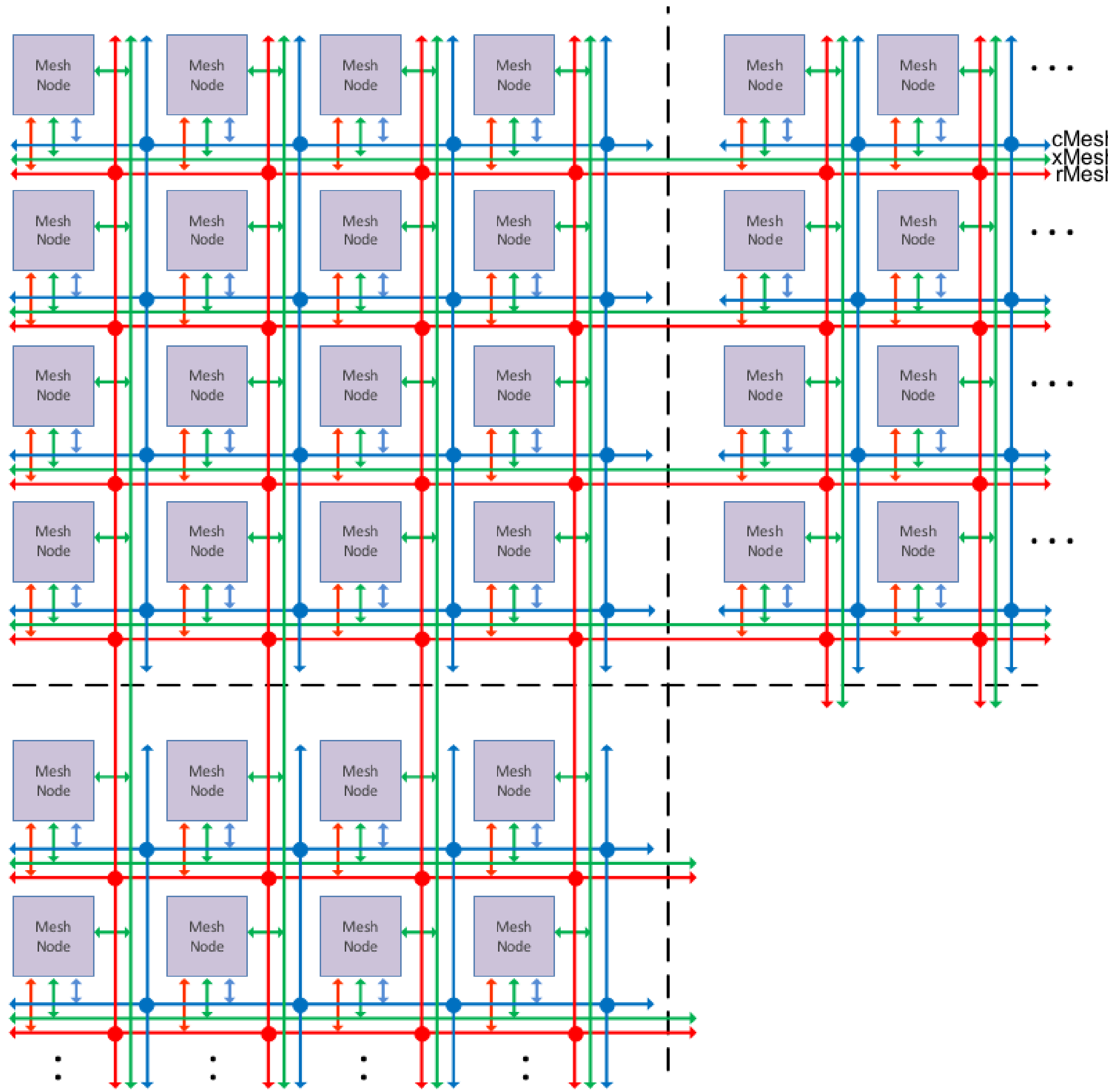
Thread 2

```
printf("good ");
printf("bye ");
```

hello world good bye	good bye hello world
hello good bye world	good hello bye world
hello good world bye	good hello world bye

Interconnect

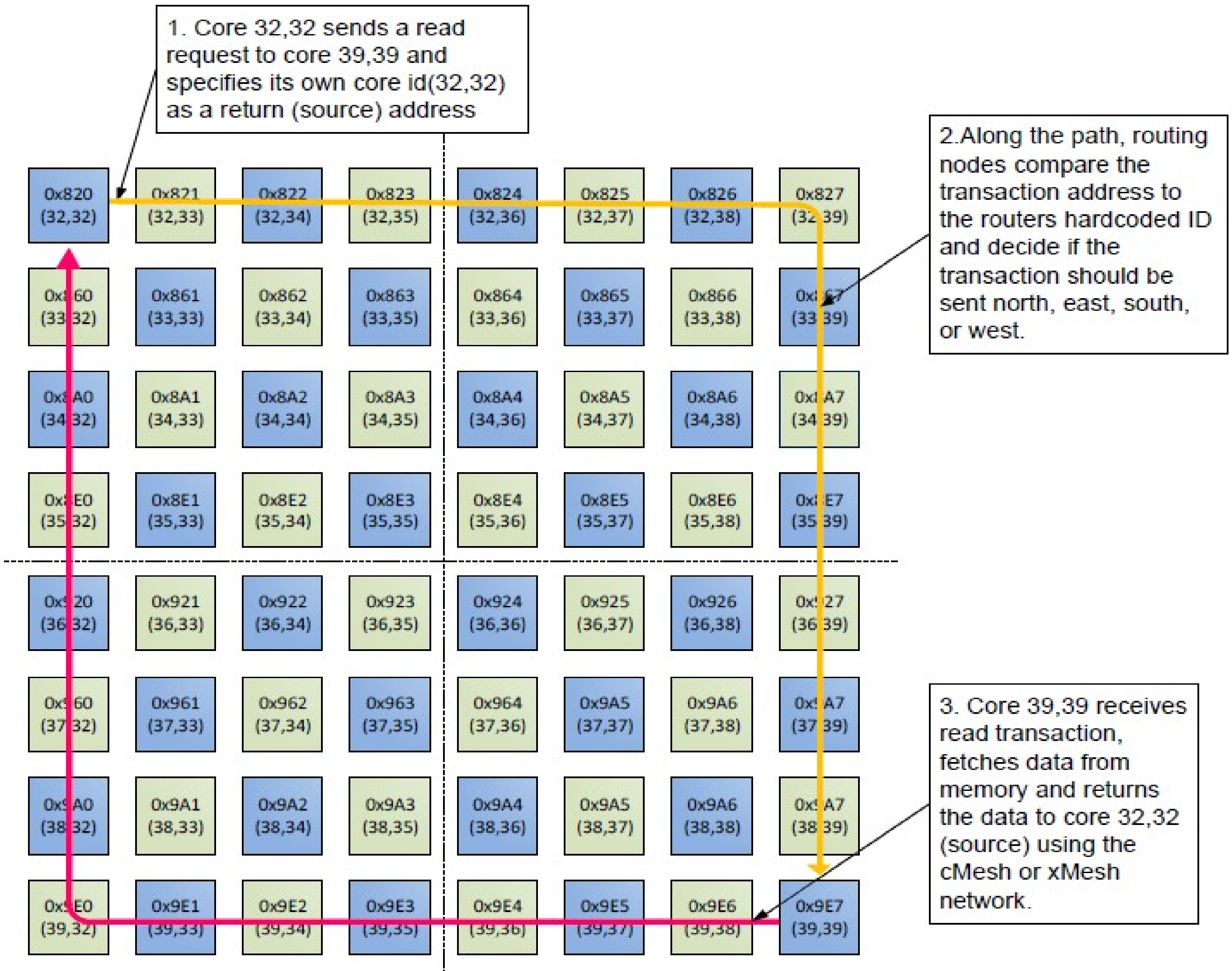
The eMesh



The eMesh

- 3 meshes
  - Read, write, off-chip
  - Fast write mesh
- Read/write to any core
- Everything handled transparently
  - But it is useful to know some of the details for performance reasons

Routing

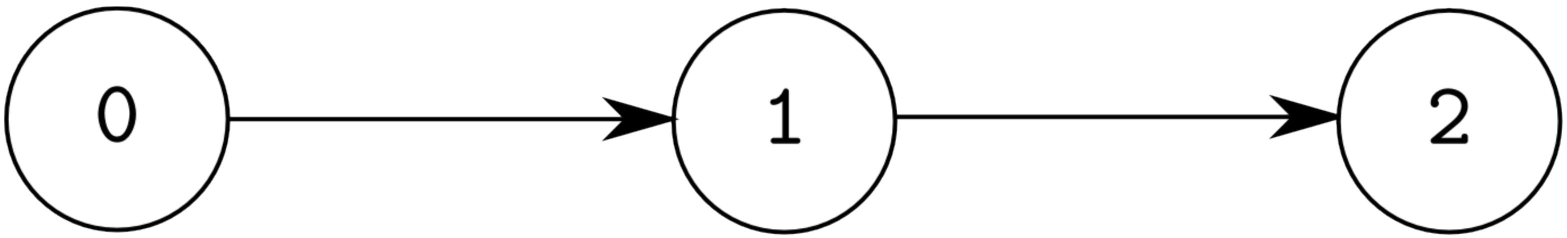

1. Core 32,32 sends a read request to core 39,39 and specifies its own core id(32,32) as a return (source) address
2.Along the path, routing nodes compare the transaction address to the routers hardcoded ID and decide if the transaction should be sent north, east, south, or west.
3. Core 39,39 receives read transaction, fetches data from memory and returns the data to core 32,32 (source) using the cMesh or xMesh network.
0x820 (32,32)
0x821 (32,33)
0x822 (32,34)
0x823 (32,35)
0x824 (32,36)
0x825 (32,37)
0x826 (32,38)
0x861 (33,33)
0x862 (33,34)
0x863 (33,35)
0x864 (33,36)
0x865 (33,37)
0x866 (33,38)
0x8A1 (34,33)
0x8A2 (34,34)
0x8A3 (34,35)
0x8A4 (34,36)
0x8A5 (34,37)
0x8A6 (34,38)
0x8E1 (35,33)
0x8E2 (35,34)
0x8E3 (35,35)
0x8E4 (35,36)
0x8E5 (35,37)
0x8E6 (35,38)
0x921 (36,33)
0x922 (36,34)
0x923 (36,35)
0x924 (36,36)
0x925 (36,37)
0x926 (36,38)
0x961 (37,33)
0x962 (37,34)
0x963 (37,35)
0x964 (37,36)
0x9A5 (37,37)
0x9A6 (37,38)
0x9A1 (38,33)
0x9A2 (38,34)
0x9A3 (38,35)
0x9A4 (38,36)
0x9A5 (38,37)
0x9A6 (38,38)
0x9E1
0x9E2
0x9E3
0x9E4
0x9E5
0x9E6
0x9E7 (39,39)

Mesh Determinism

- Relaxed consistency
- Be careful when writing to other cores!
- Order of writing to 2 different cores is not deterministic – race conditions and deadlock
- **Writing to a remote core, then attempting to read back the value is not deterministic**

EMBECOSM®

Safe Example



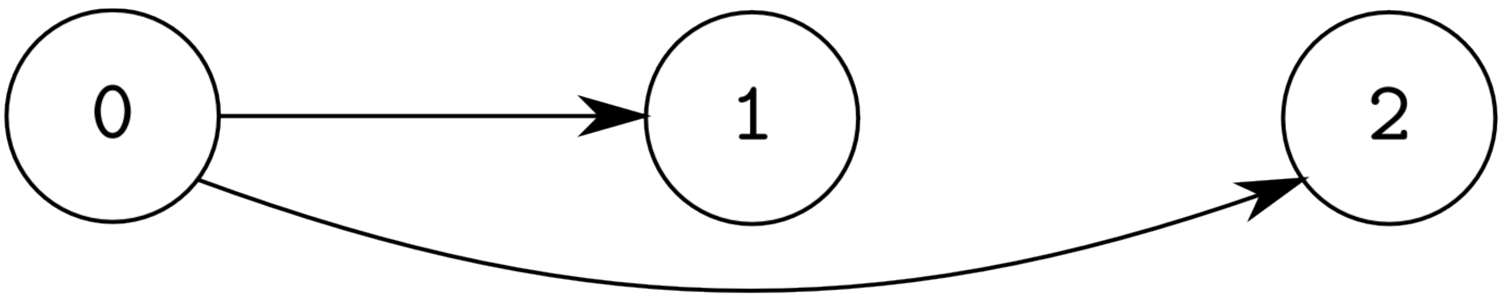
send to core 1

receive from core 0
print “hello”
send to core 2

receive from core 1
print “world”

hello world

Unsafe example



send to core 1
send to core 2

receive from core 0
print “hello”

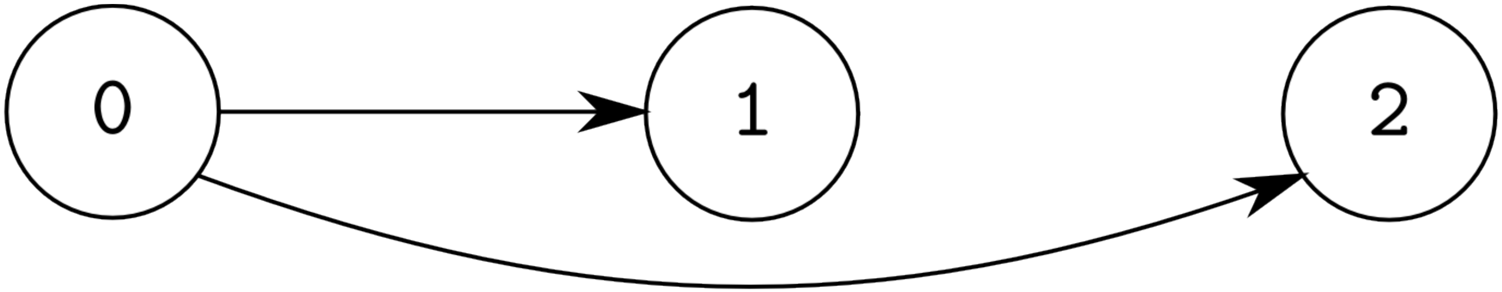
receive from core 1
print “world”

hello world

Lucky

EMBECOSM®

Unsafe example



send to core 1
send to core 2

receive from core 0
print “hello”

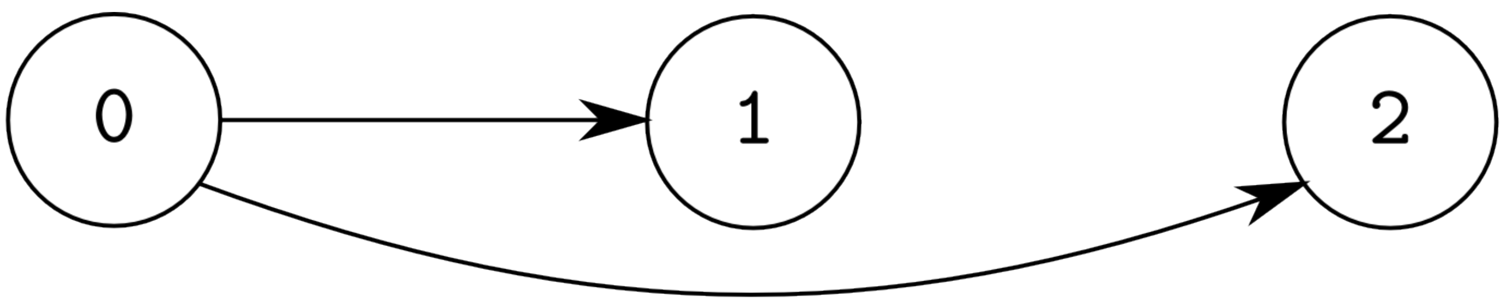
receive from core 1
print “world”

world hello

Unlucky

Make it safe

- How do we make it safe?
  - Synchronisation



send to core 1
receive from core 1
send to core 2

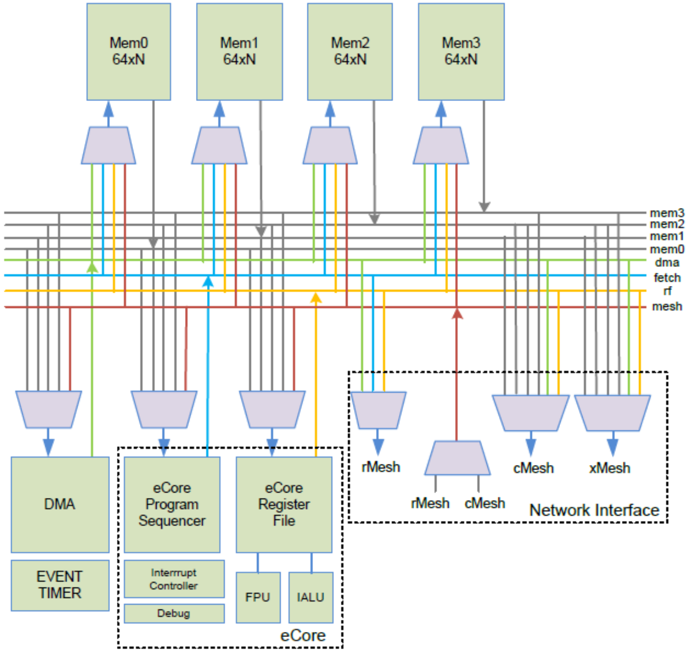
receive from core 0
print “hello”
send to core 0

receive from core 1
print “world”

hello world

Make it fast

- Write mesh is non-blocking
- Try to structure programs and algorithms to only write to *non-local* memory
- Read and write from *local* memory is fast



EMBECOSM®

A simple example - slow

Running on core 0

```
int a[10], b[10];      // On core 0
int c;                 // On core 1

for(i = 0, c = 0; i < 10; ++i)
    c += a[i] * b[i];
```

Note – without synchronisation this program is also non-deterministic

A simple example - fast

Running on core 0

```
int a[10], b[10];       // On core 0
int c_temp;             // On core 0
int c;                  // On core 1

for(i = 0, c_temp = 0; i < 10; ++i)
    c_temp += a[i] * b[i];

c = c_temp;
```

Fortunately this program is deterministic

EMBECOSM®

A tricky example

Running on core 0

```
int a[10], b[10];   // On core 0
int c[10];          // On core 1

for(i = 0; i < n; ++i)
    c[i] += a[i] * b[i];
```

EMBECOSM®

Another Solution?

Core 0

```
int a[10], b[10];
int *temp;

temp = &partial;

for(i = 0; i < 10; ++i)
{
    temp[i] = a[i] * b[i];
    sync();
}
```

Core 1

```
int c[10];
int partial[10];


for(i = 0; i < 10; ++i)
{
    sync();
    c[i] += partial[i];
}
```

Is it faster?

Conclusion/Tips

- Many different paradigms for parallelism
- Careful of resource sharing
- Race conditions
  - Unique challenges with accessing remote cores
- Writing is faster
  - But it's challenging to pick the best way of doing this

More Info

Epiphany architecture reference manual

http://www.adapteva.com

http://www.parallella.org

http://www.embecosm.com

Questions?