Two High-Throughput Architectures and the Compilers Who Love Them

A Talk for 15745-s07 by

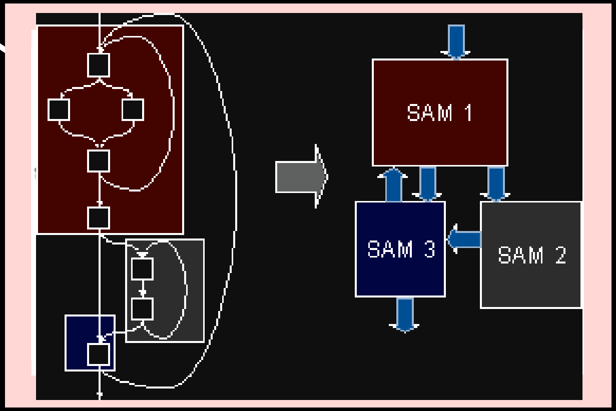
Ronit Slyper - rys@cs.cmu.edu
Jim McCann - jmccann@cs.cmu.edu

(with diagrams borrowed from IBM and NVIDIA)

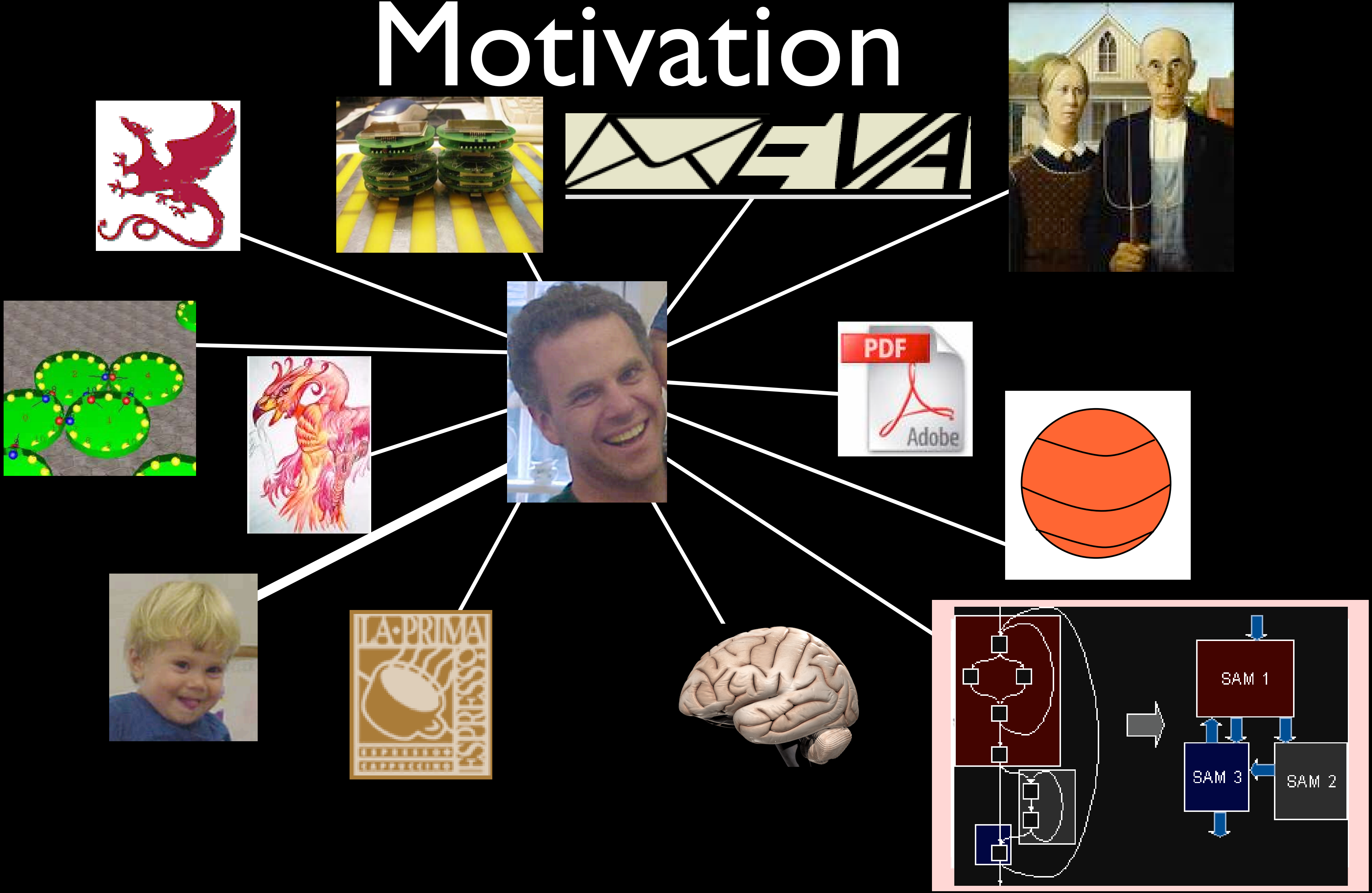
Motivation

Motivation

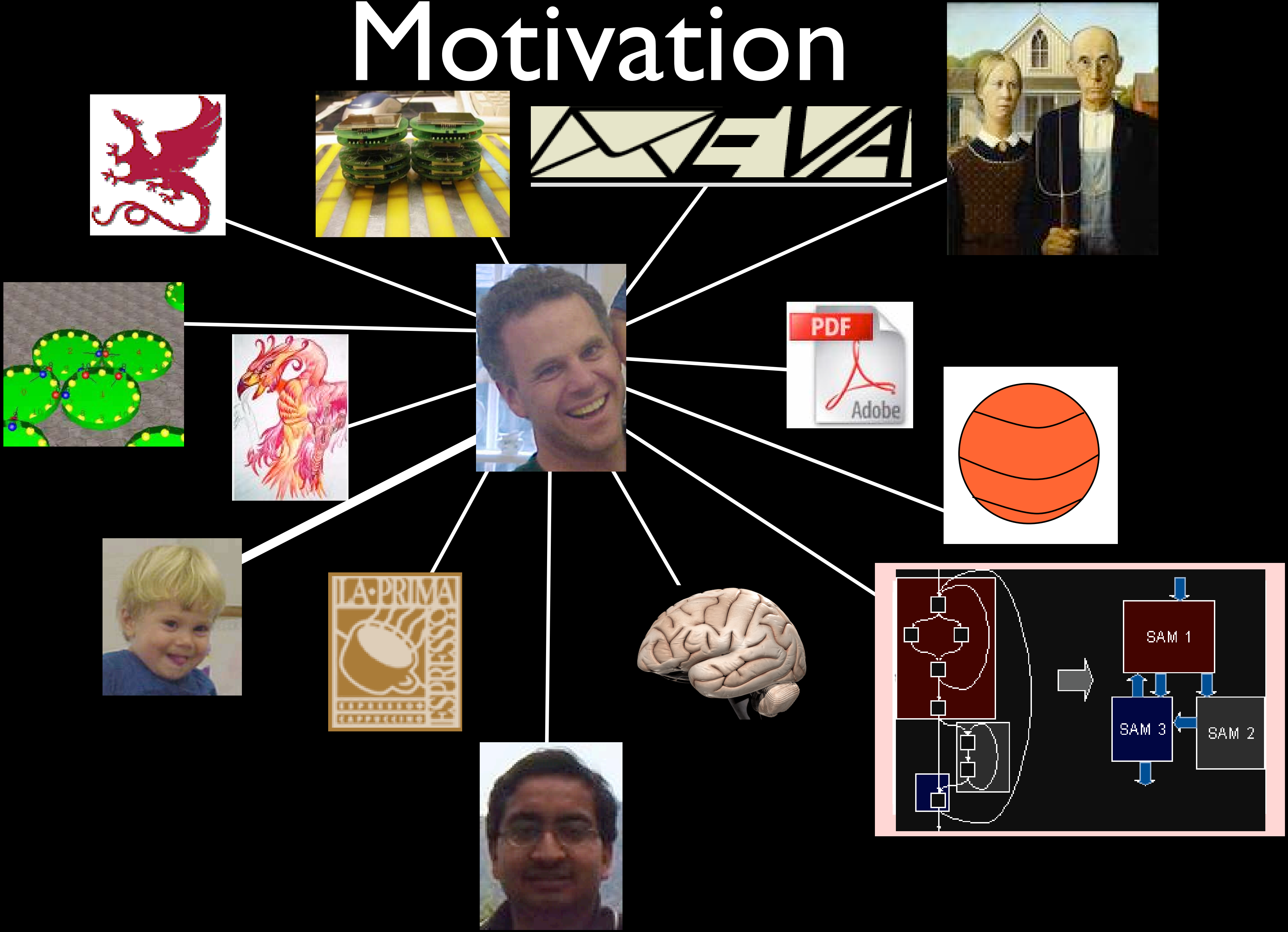
Motivation



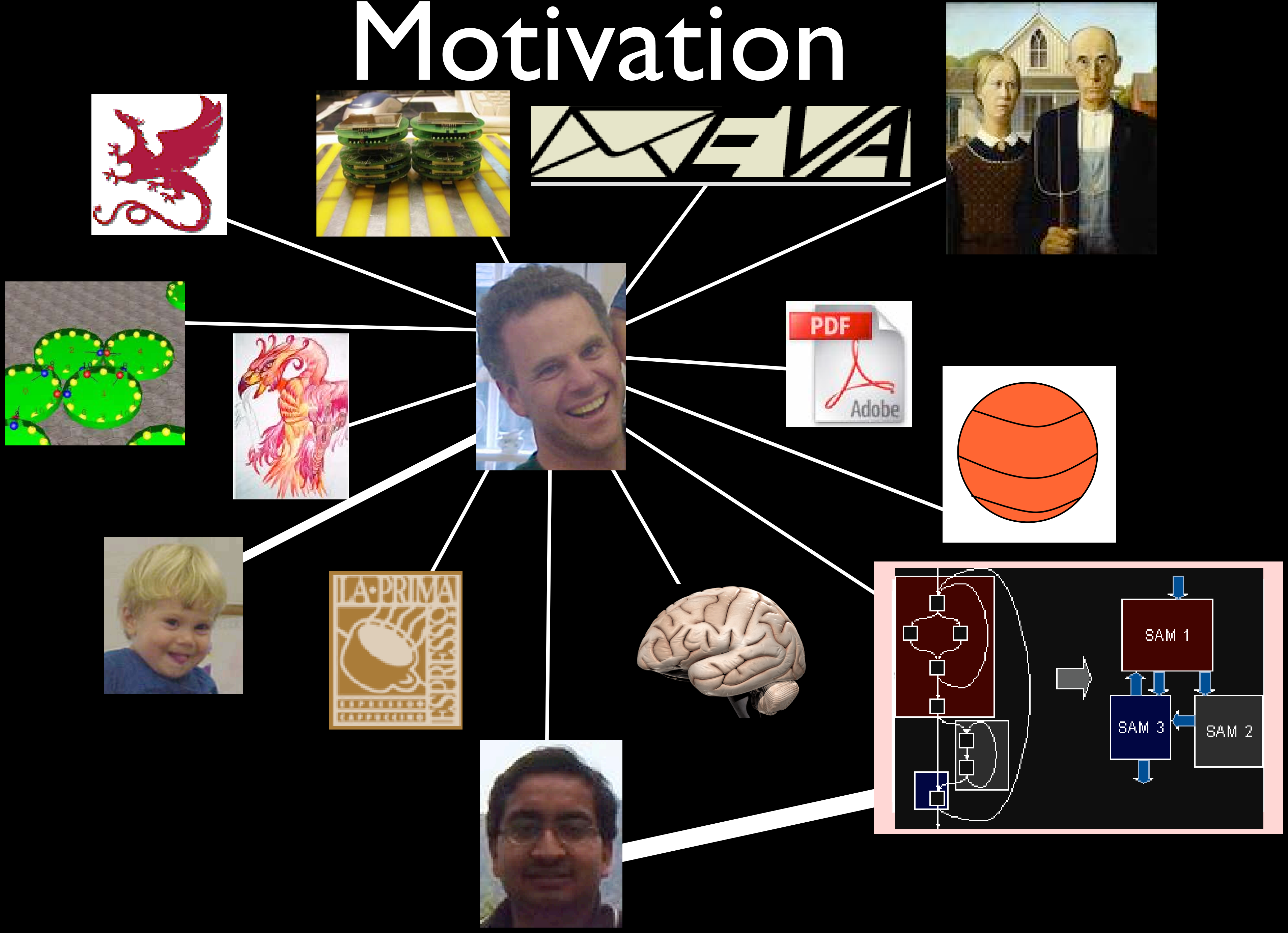
Motivation



Motivation



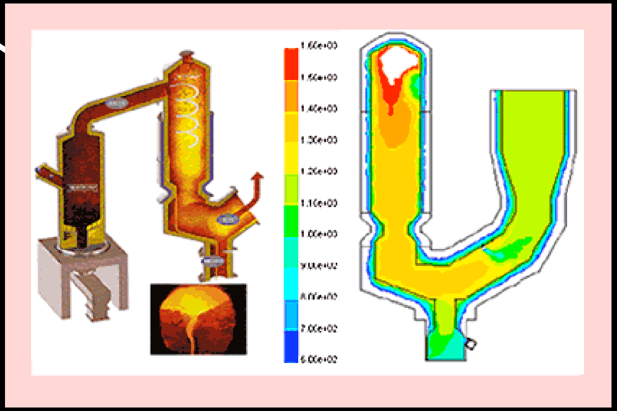
Motivation



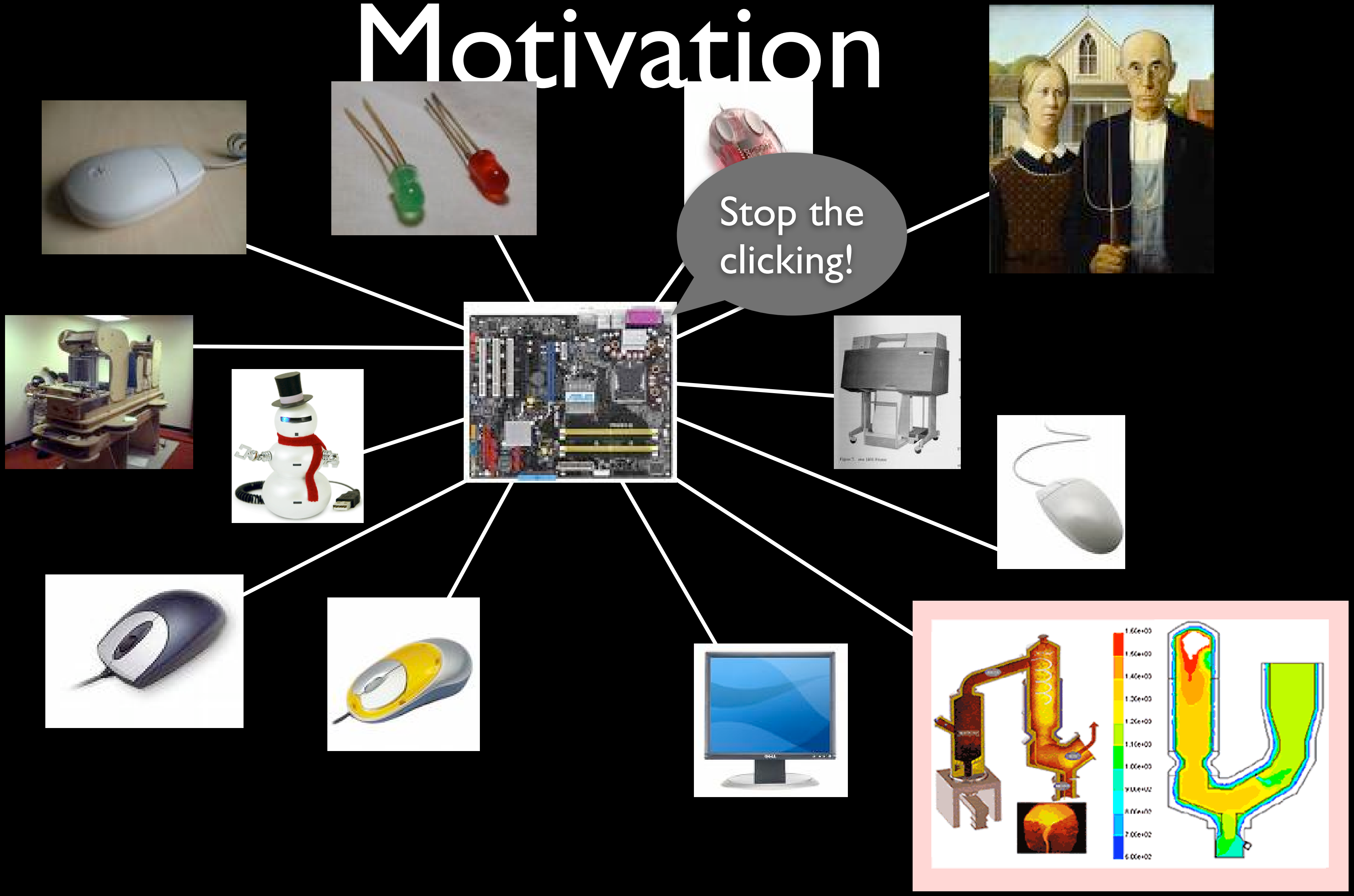
Motivation

Motivation

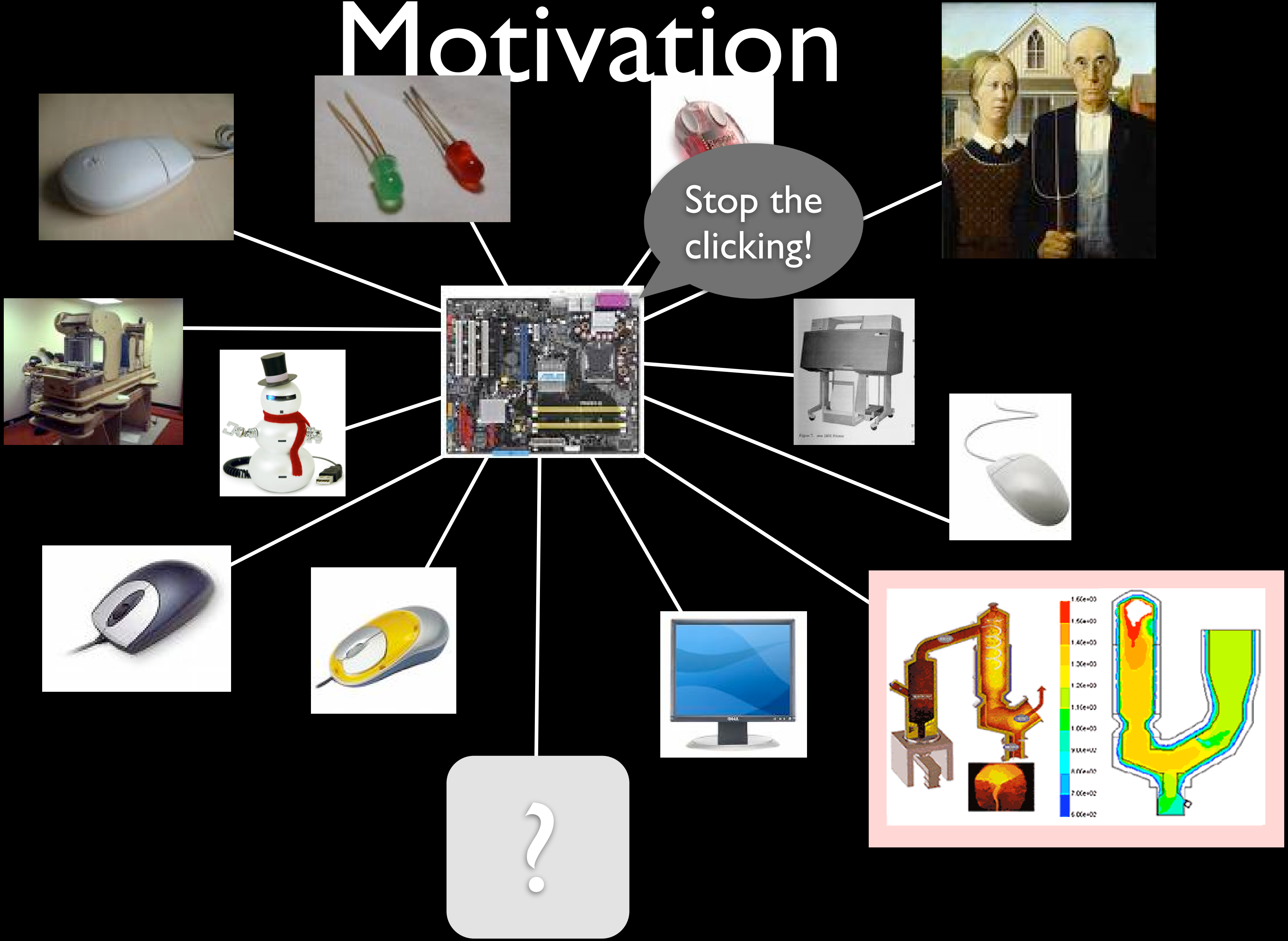
Motivation


1.60e+03
1.50e+03
1.40e+03
1.30e+03
1.20e+03
1.10e+03
1.00e+03
9.00e+02
8.00e+02
7.00e+02
6.00e+02

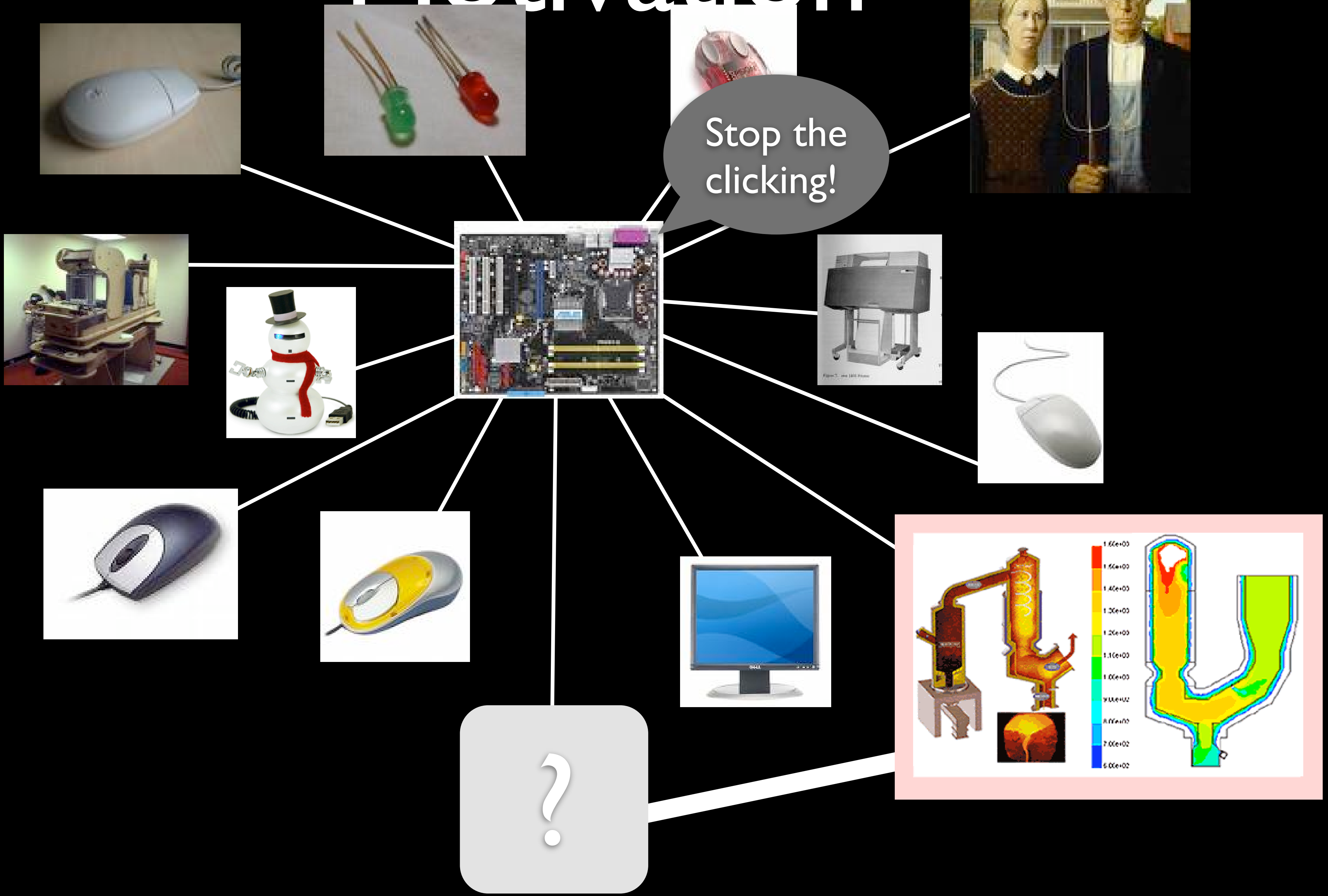
Motivation



Motivation

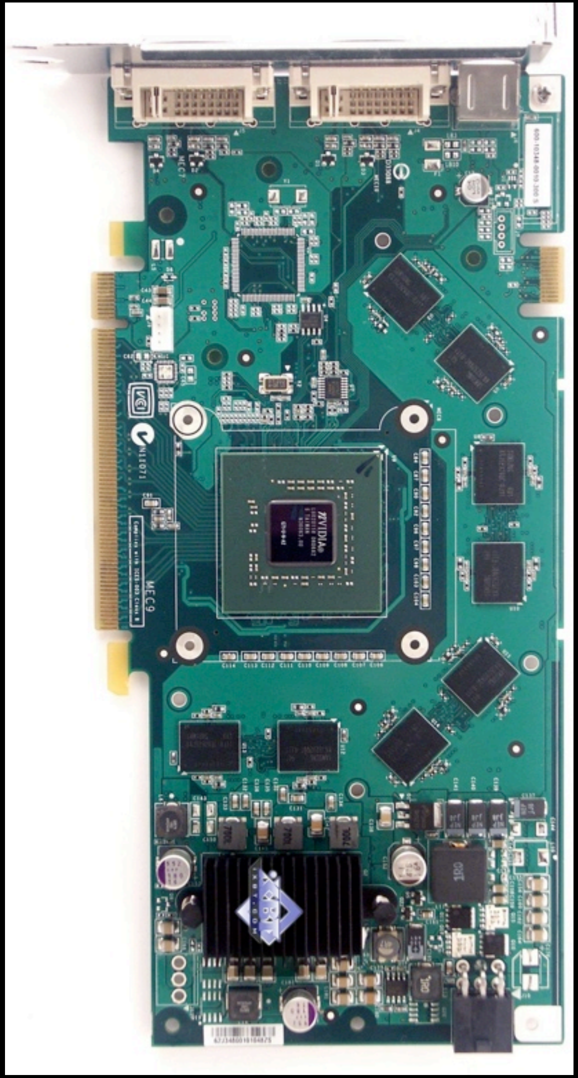

Stop the clicking!
1.60e+03
1.50e+03
1.40e+03
1.30e+03
1.20e+03
1.10e+03
1.00e+03
9.00e+02
8.00e+02
7.00e+02
6.00e+02
?

Motivation



Two Architectures

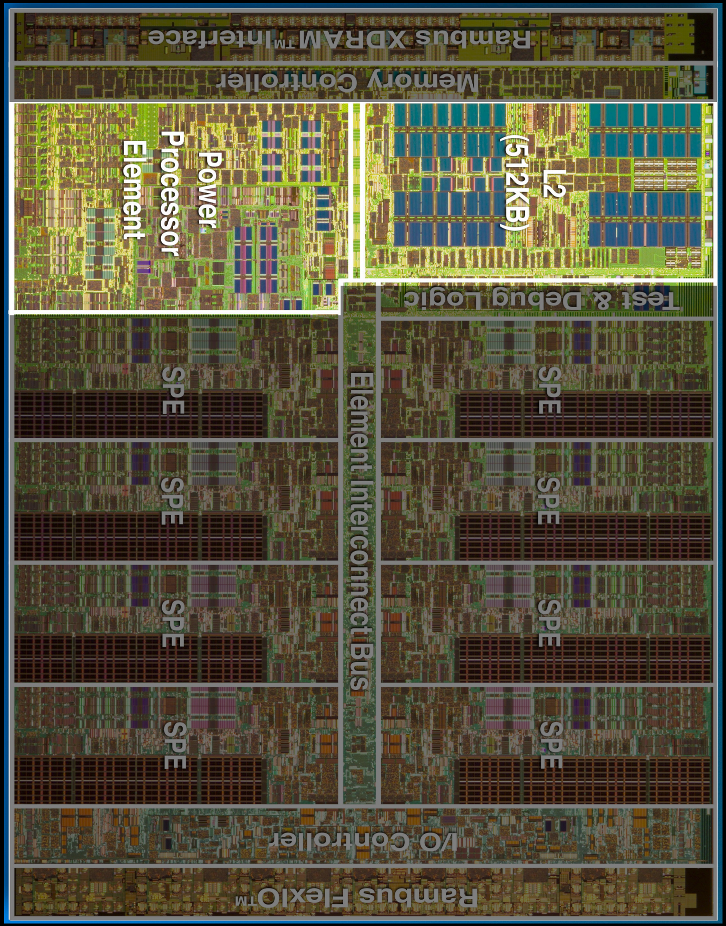
IBM Cell BE



NVIDIA G8x

The “ ” Portion

On-die PowerPC Processor

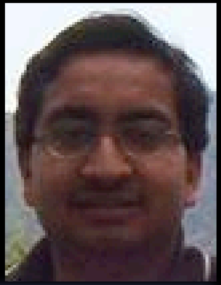


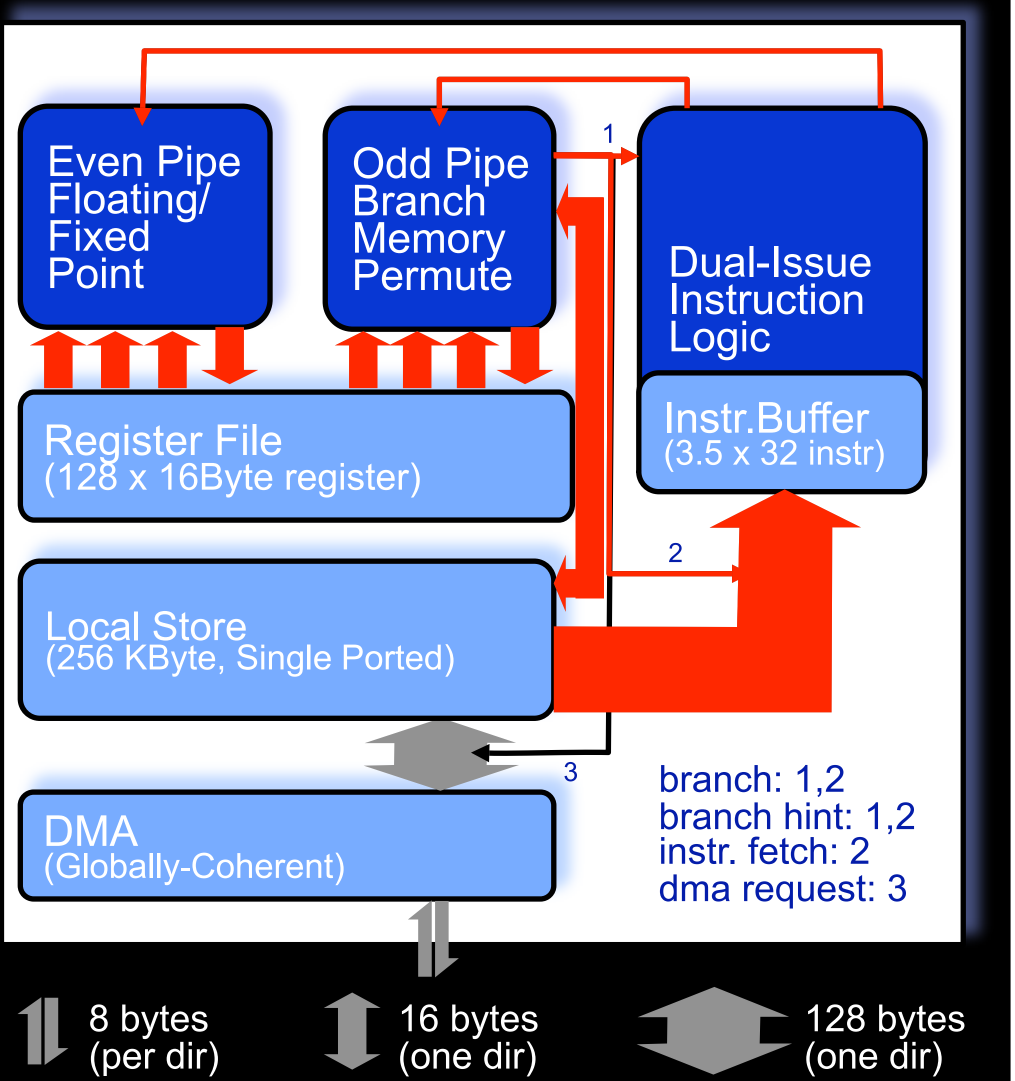
IBM Cell BE

Your desktop’s main CPU

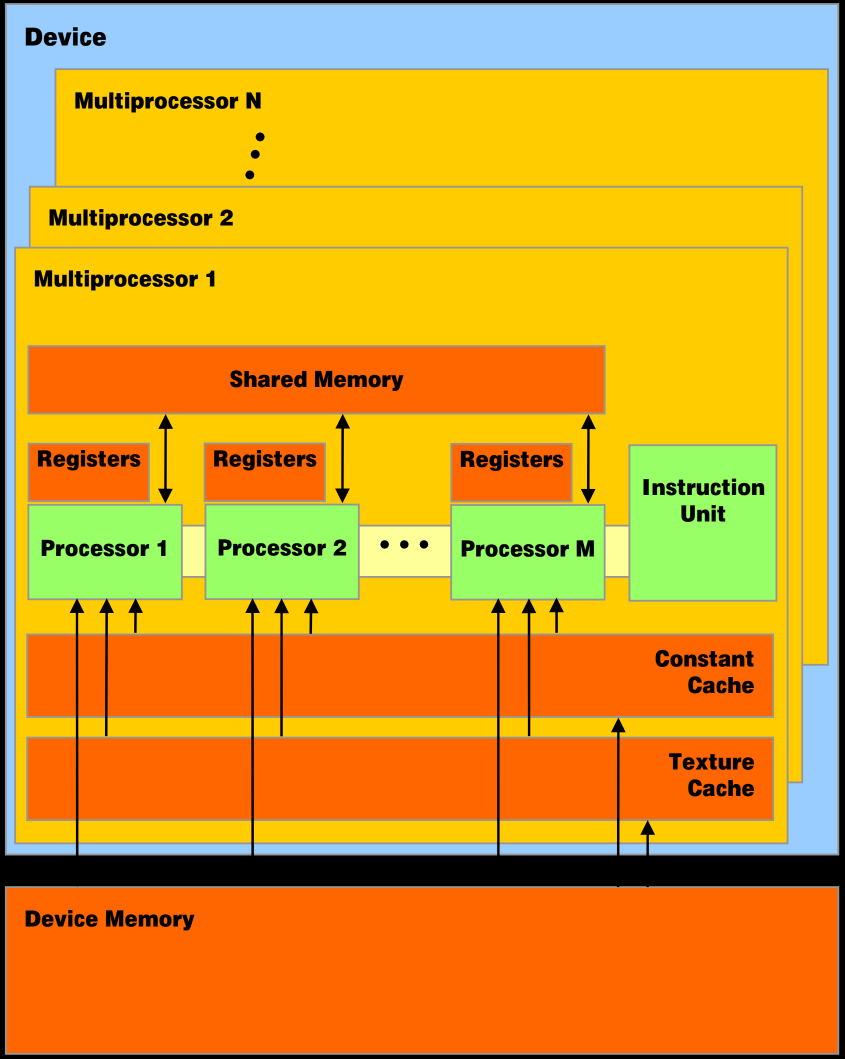
NVIDIA G8x

The “ ” Portion





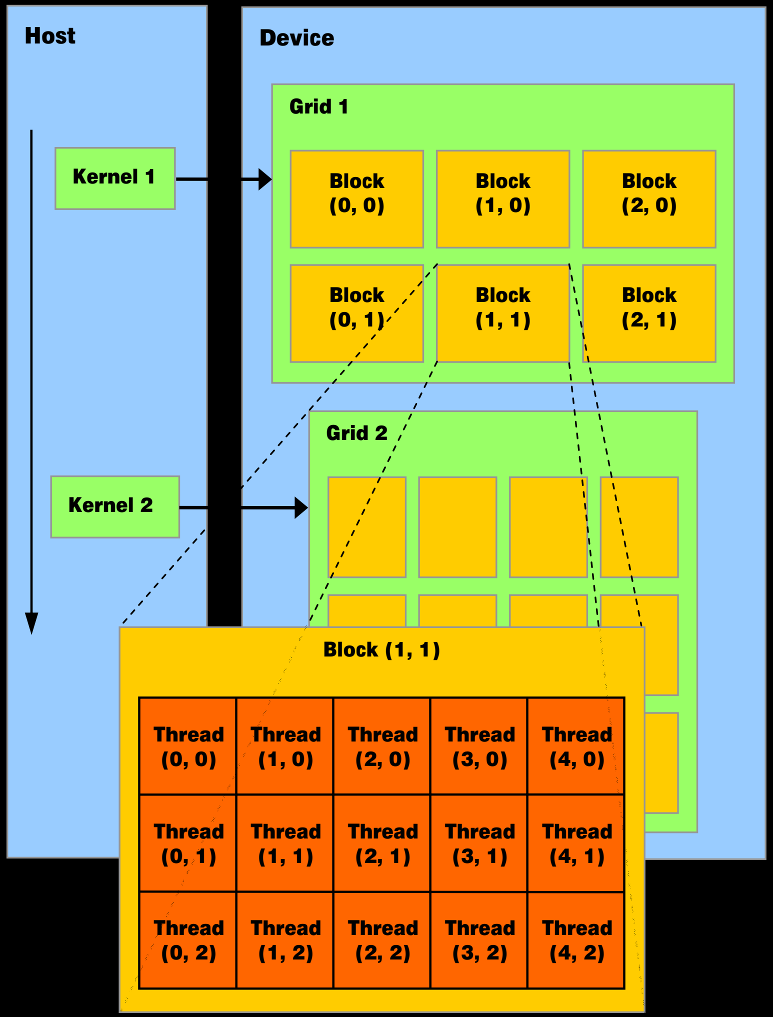
IBM Cell BE



NVIDIA G8x

Operation

“Blocks of Threads”



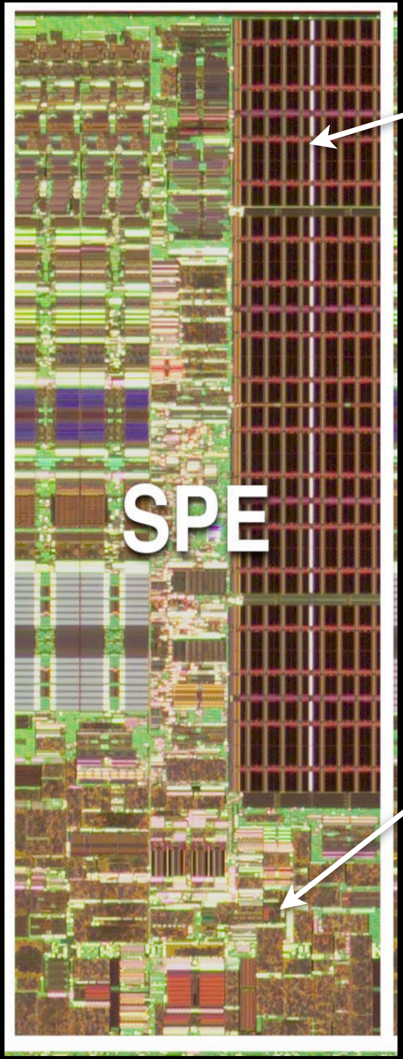
Single-threaded

Dual-issue

“Normal”

IBM Cell BE | NVIDIA G8x

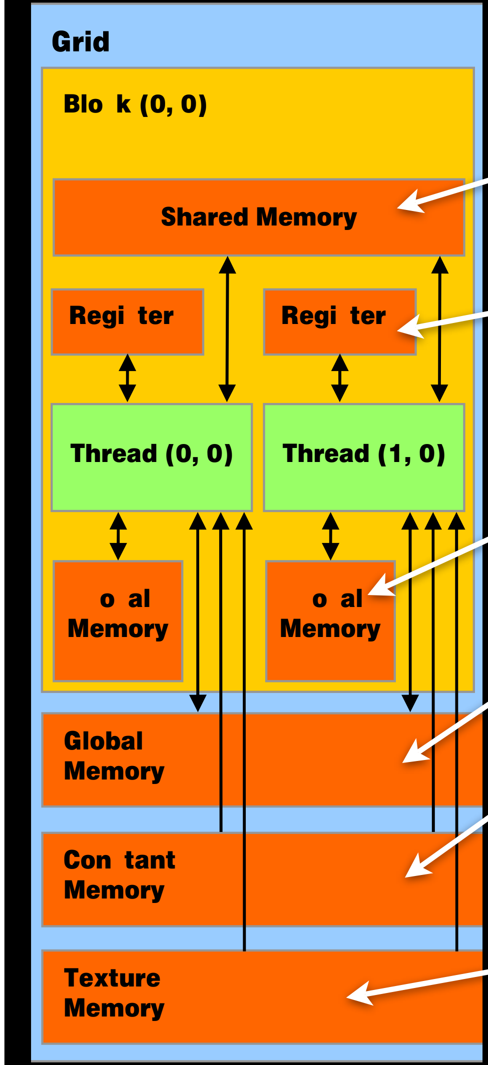
Memory



Local Store (256K)

DMA Engine to copy data to and from the local store

IBM Cell BE



16KB / 16

?? / blocks

Abstraction of shared

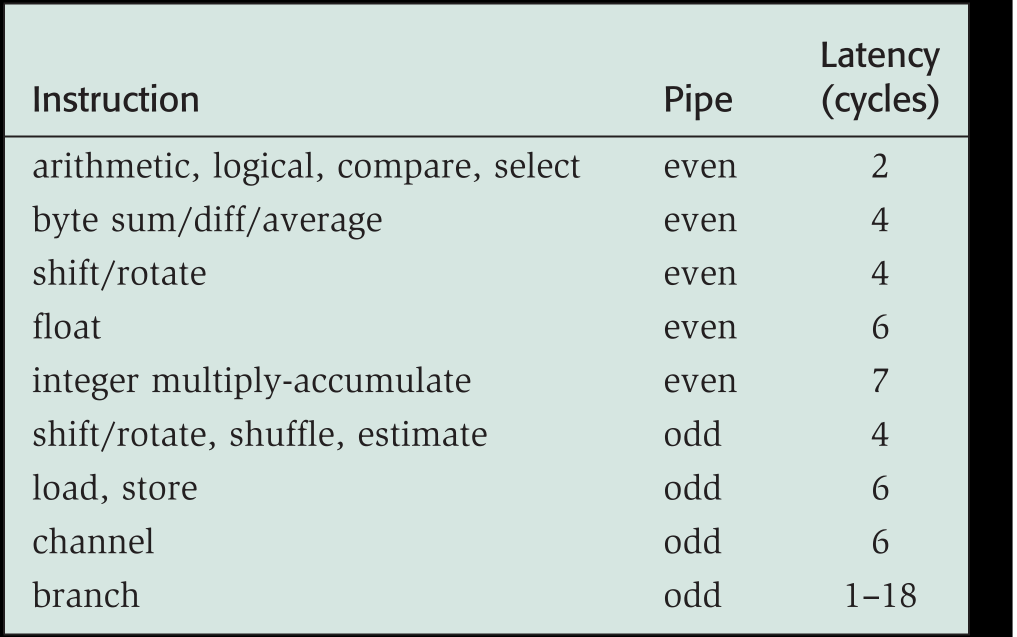
not cached

64KB, 8KB cache

abstraction of global w/ spatial cache and filtering.

NVIDIA G8x

Instruction Latencies



Instruction	Pipe	Latency (cycles)
arithmetic, logical, compare, select	even	2
byte sum/diff/average	even	4
shift/rotate	even	4
float	even	6
integer multiply-accumulate	even	7
shift/rotate, shuffle, estimate	odd	4
load, store	odd	6
channel	odd	6
branch	odd	1–18

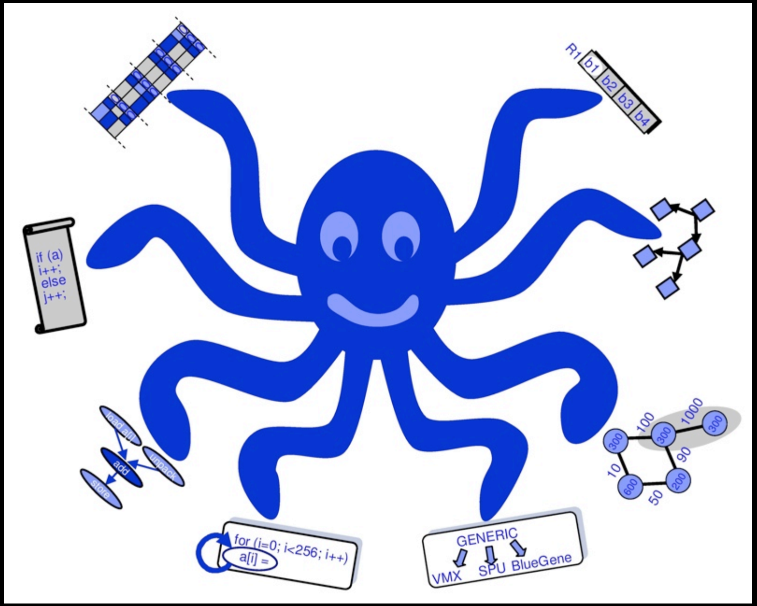
IBM Cell BE

most float, int ops	2
float inv, 1/sqrt, log	8
int 32-bit mul	8
int 24-bit mul	2
int div, mod	“slow”
sin, cos, exp, sqrt	16
float div	18/10
local load/store	2
global load/store	200+
sync	2

Note: latencies sometimes hidden by swapping thread blocks

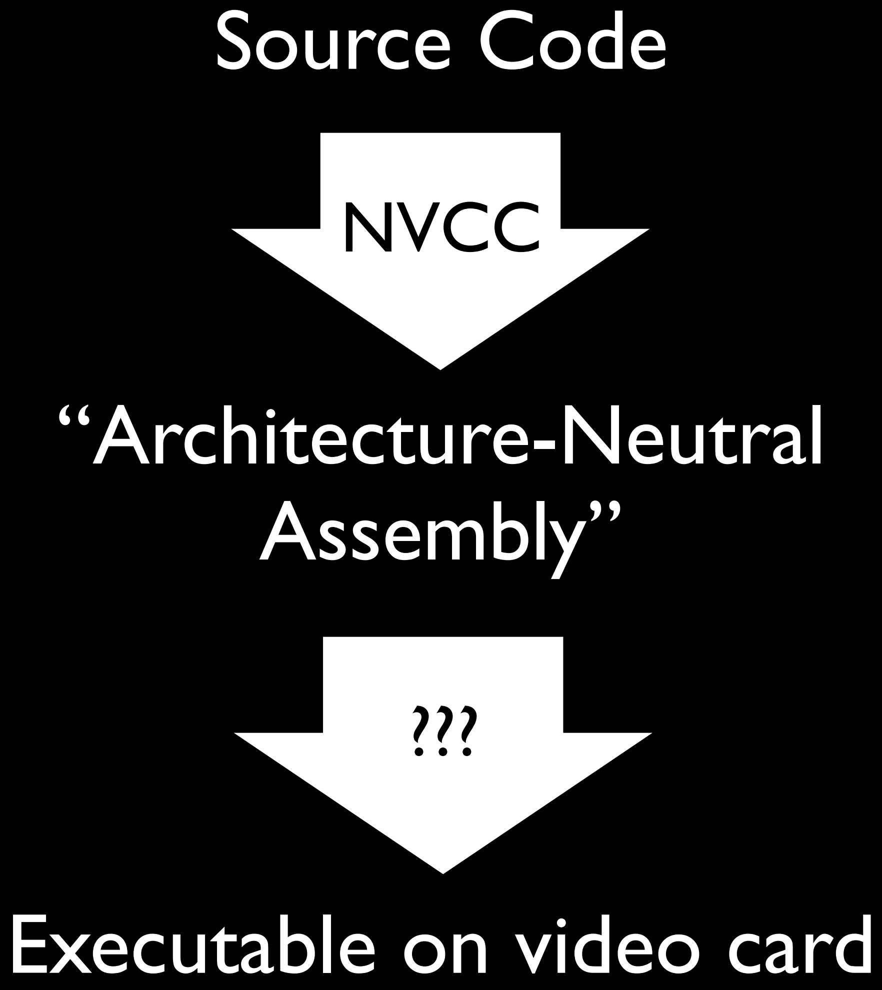
NVIDIA G8x

Two-ish Compilers



- Single Source to PPE + SPE
- Complicated optimizations

IBM Cell BE


Source Code

NVCC

"Architecture-Neutral Assembly"

???

Executable on video card

NVIDIA G8x

The Programmer’s Job

(Optionally) indicate parallelism.

IBM Cell BE

Explicitly specify and coordinate threads (no compiler help).

NVIDIA G8x

The Compiler’s Job

IBM Cell BE | NVIDIA G8x

The Compiler's Job

IBM Cell BE

The Compiler’s Job

Provide a “Single Program” Abstraction

Deal with the Local Store:

- fit code on SPE
- handle global memory access
- prevent instruction starvation
- hide memory latency

IBM Cell BE

The Compiler's Job

Provide a "Single Program" Abstraction

Deal with the Local Store:

- fit code on SPE
- handle global memory access
- prevent instruction starvation
- hide memory latency

Vectorization
Scalar Variable Overhead

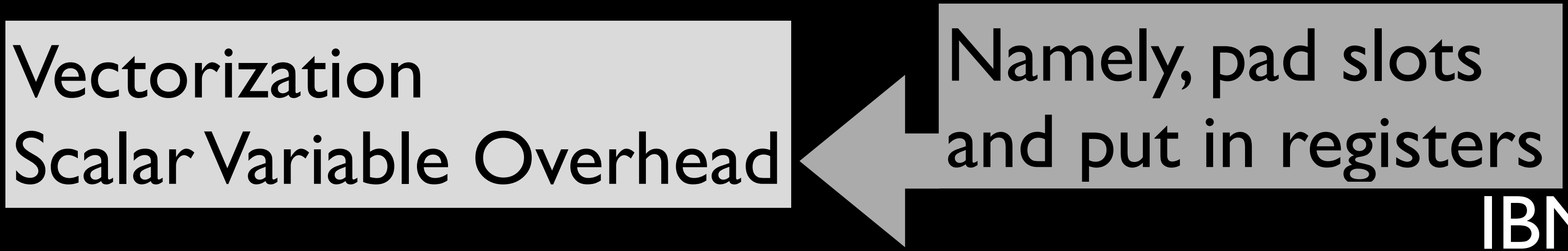
IBM Cell BE

The Compiler’s Job

Provide a “Single Program” Abstraction

Deal with the Local Store:

- fit code on SPE
- handle global memory access
- prevent instruction starvation
- hide memory latency



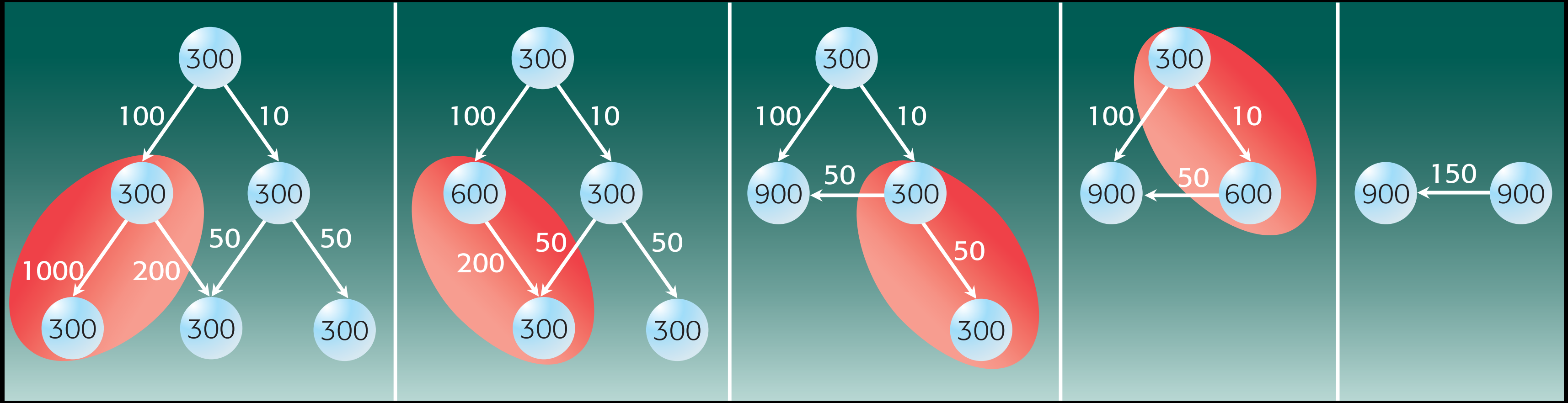
Vectorization
Scalar Variable Overhead

Namely, pad slots and put in registers

IBM Cell BE

Code Partitioning

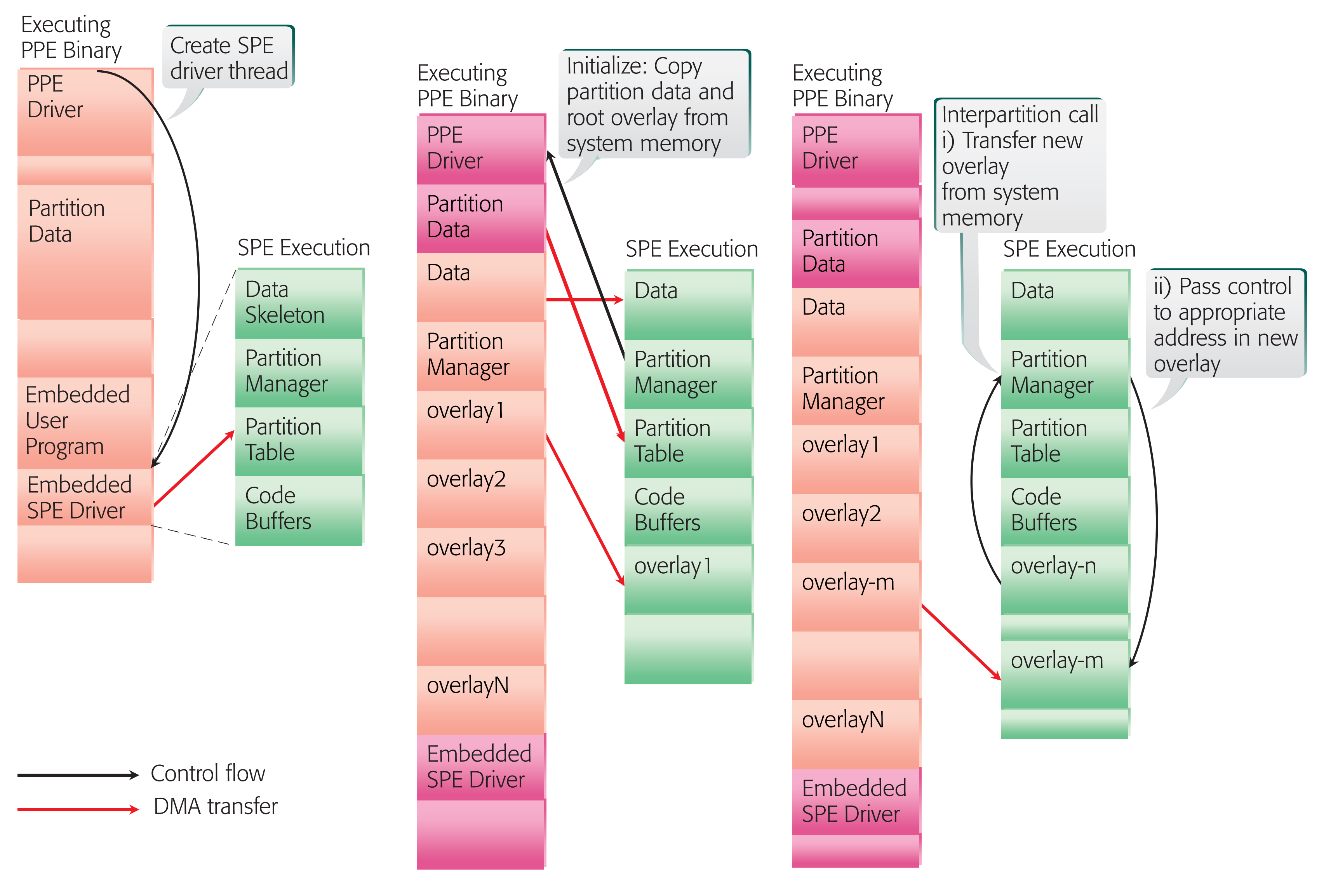
Greedily collapse functions into same partition (when they fit) -- minimize inter-partition calls.



Call Graph -- Edges are call frequencies

IBM Cell BE

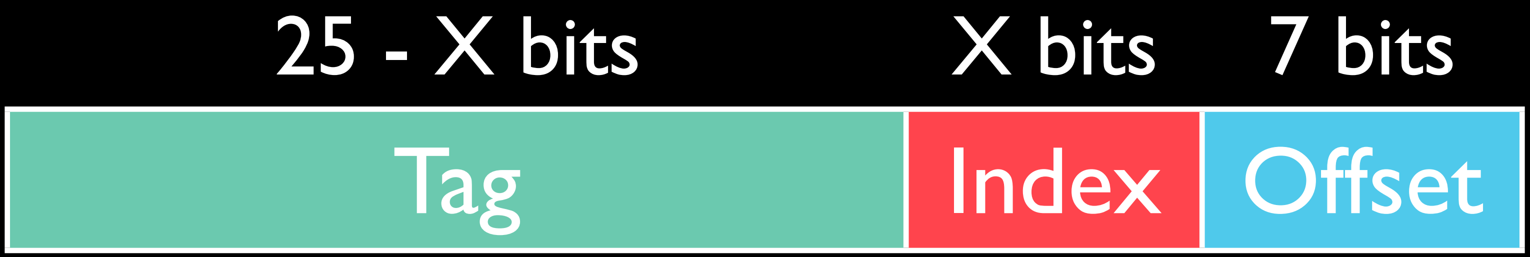
Code Partitioning (runtime)



IBM Cell BE

Software Cache

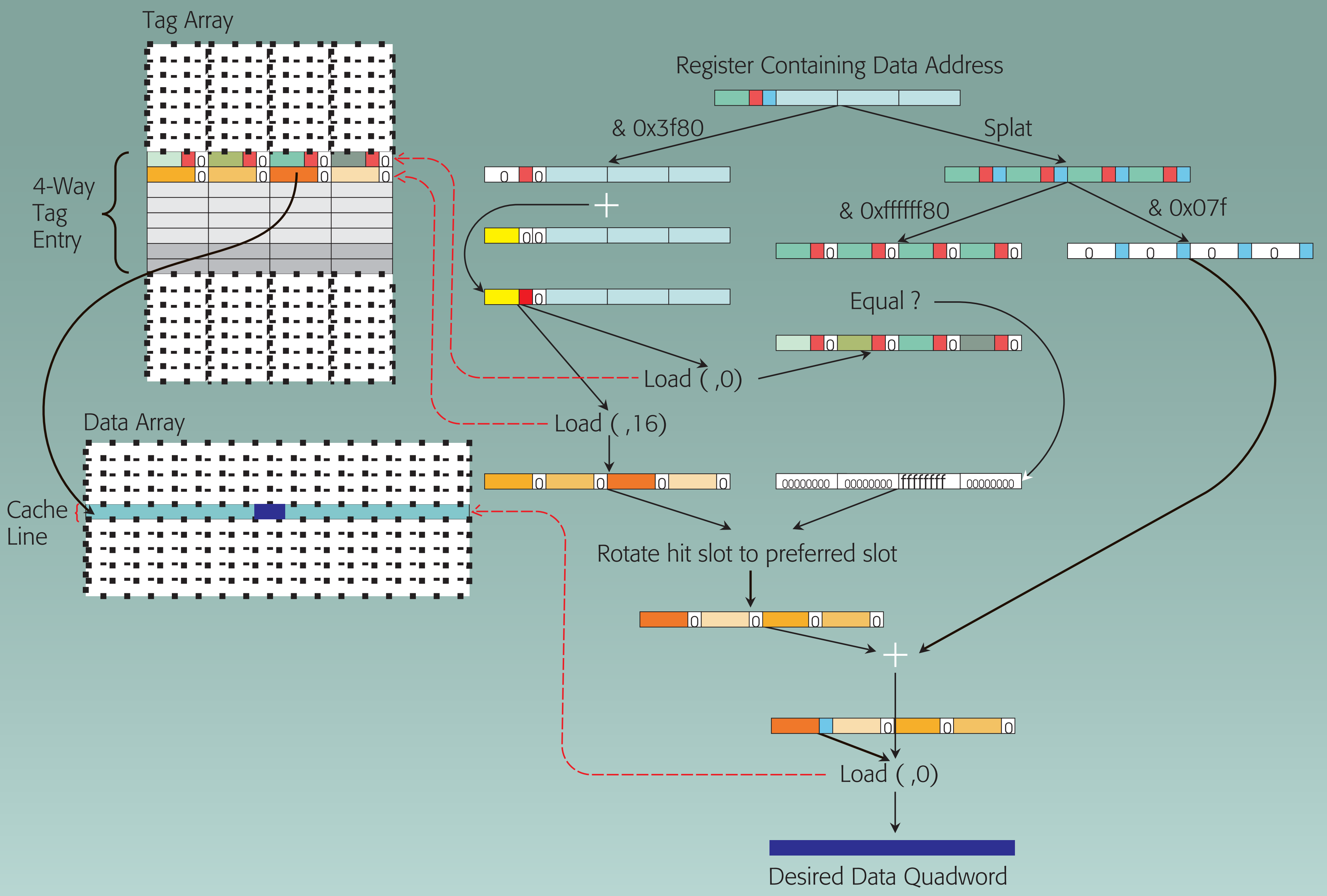
Does the hard case right by emulating a data cache.
(4-way set associative, 128-byte lines)



25 - X bits	X bits	7 bits
Tag	Index	Offset

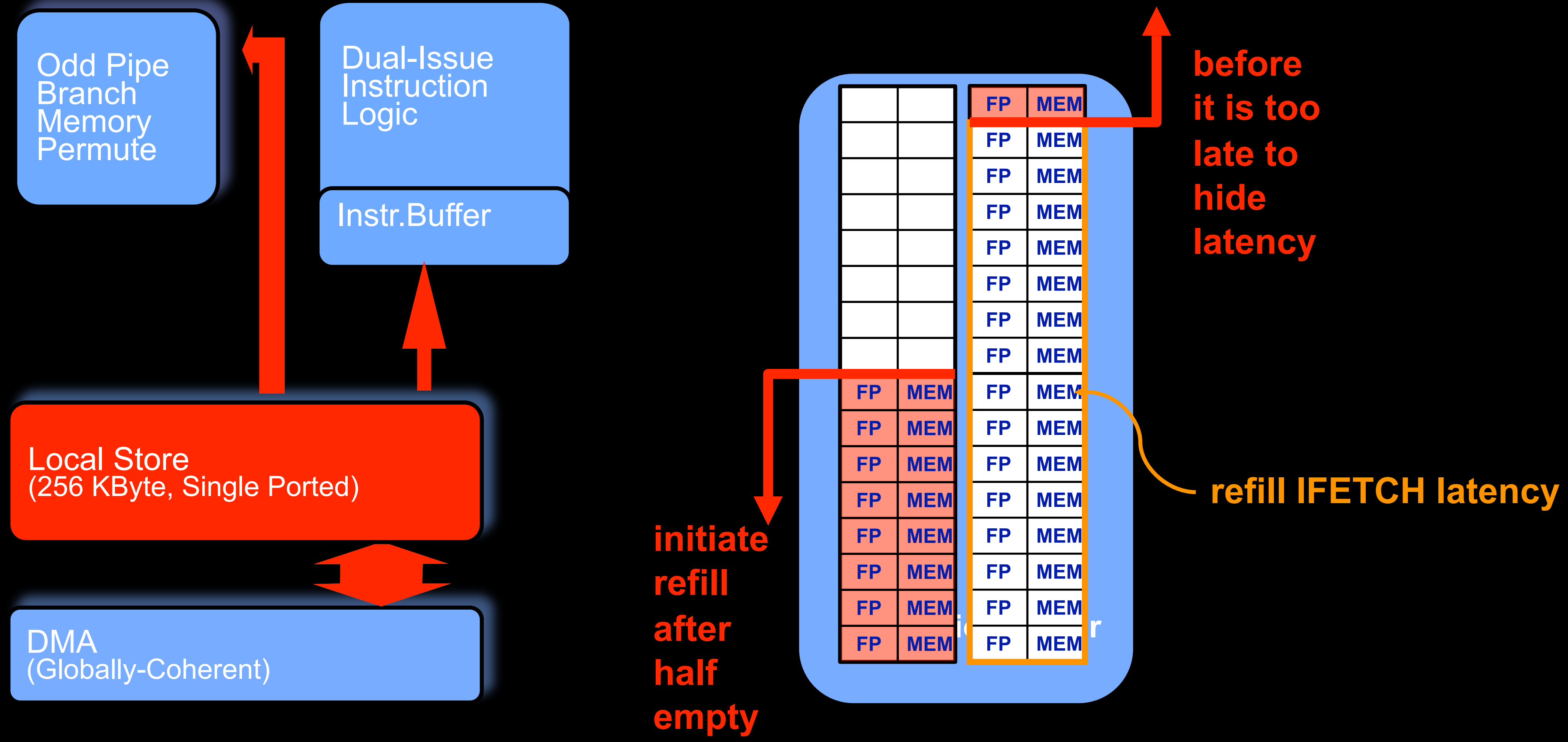
32-bit address

IBM Cell BE



🟩 High tag bits for hit
□ High tag bits of other lines for this tag
🟥 Tag index
🟦 Offset in line
□ Don’t care
🟨 Address of tag array (high bits)
🟧 Address of hit cache line
🟧 Addresses of other lines for this tag
□ Dirty bits in tag entry
▨ Unused space in tag entry
→ Register operation result
⇢ Memory location for load

Instruction Starvation



Odd Pipe
Branch
Memory
Permute

Dual-Issue
Instruction
Logic

Instr.Buffer

Local Store
(256 KByte, Single Ported)

DMA
(Globally-Coherent)

		FP	MEM
		FP	MEM
		FP	MEM
		FP	MEM
		FP	MEM
		FP	MEM
		FP	MEM
		FP	MEM
FP	MEM	FP	MEM
FP	MEM	FP	MEM
FP	MEM	FP	MEM
FP	MEM	FP	MEM
FP	MEM	FP	MEM
FP	MEM	FP	MEM
FP	MEM	FP	MEM
FP	MEM	FP	MEM

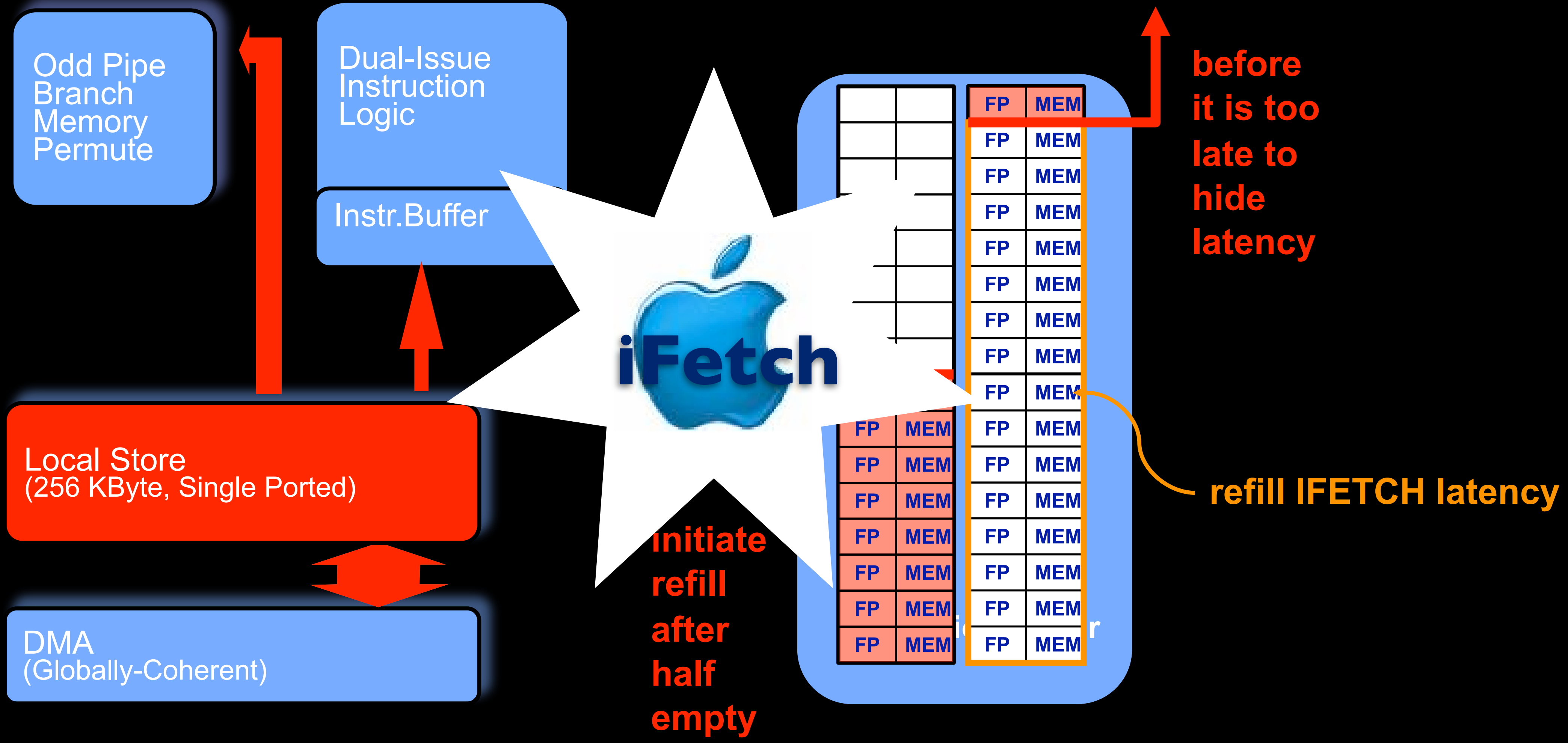
before
it is too
late to
hide
latency

refill IFETCH latency

initiate
refill
after
half
empty

IBM Cell BE

Instruction Starvation



Odd Pipe
Branch
Memory
Permute

Dual-Issue
Instruction
Logic

Instr.Buffer

Local Store
(256 KByte, Single Ported)

DMA
(Globally-Coherent)

iFetch

initiate refill after half empty

before it is too late to hide latency

refill IFETCH latency

IBM Cell BE

Hiding Memory Latency

Simple: Allocate local variables locally.

Tricky: Use array tiling.

IBM Cell BE

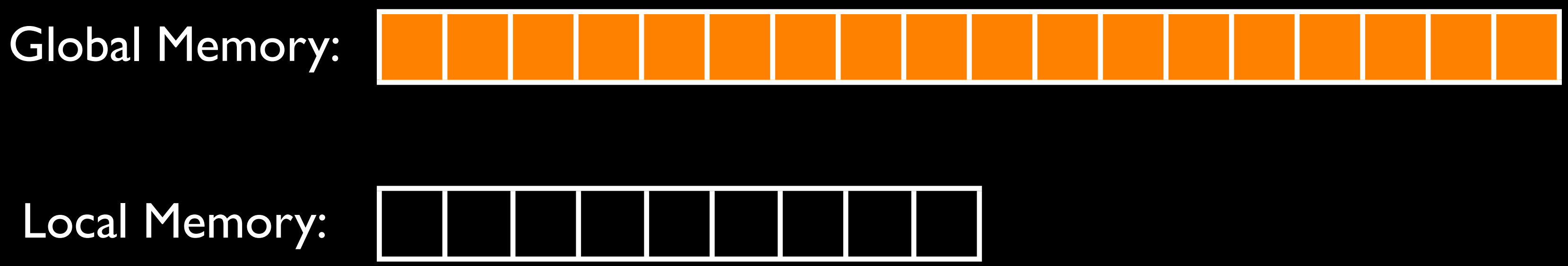
Array Tiling By Example

“Sum A”

```
sum = 0;
for (i = 0; i < 100; ++i) {
     sum += A[i];
}
```

IBM Cell BE

Before:



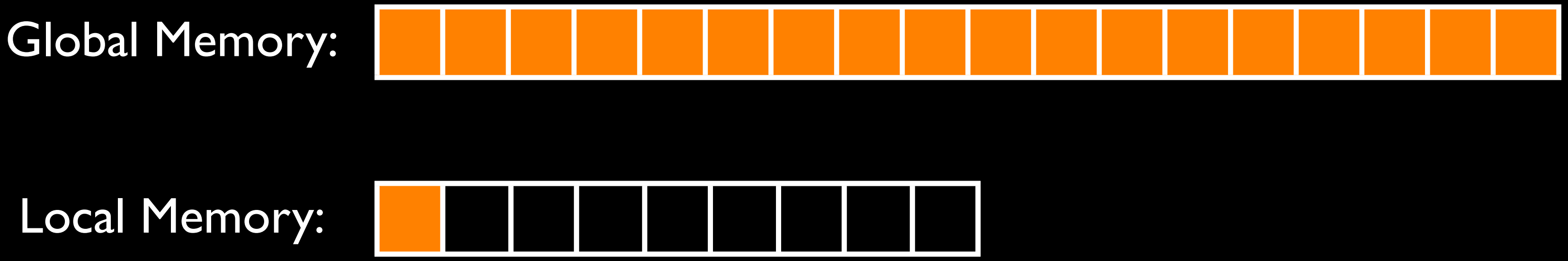
Global Memory:

Local Memory:

```
sum = 0;
for (i = 0; i < 100; ++i) {
  Copy A[i] to Local[0];
  sum += Local[0];
}
```

IBM Cell BE

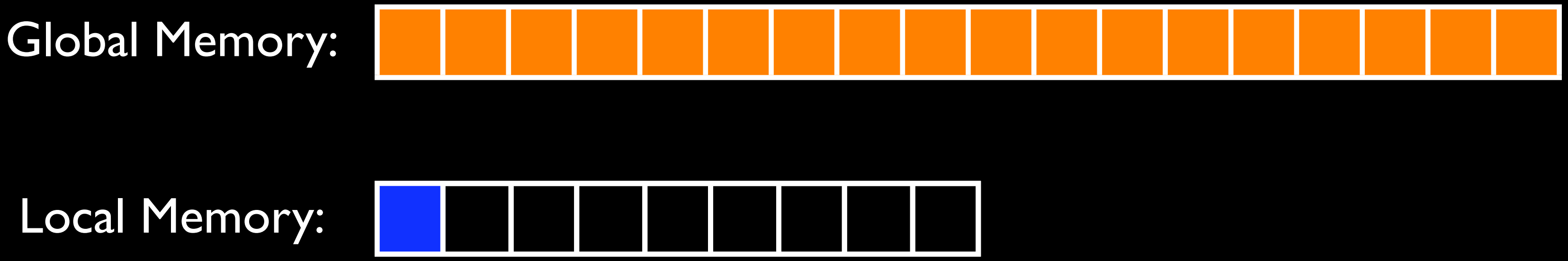
Before:



```
sum = 0;
for (i = 0; i < 100; ++i) {
  Copy A[i] to Local[0];
  sum += Local[0];
}
```

IBM Cell BE

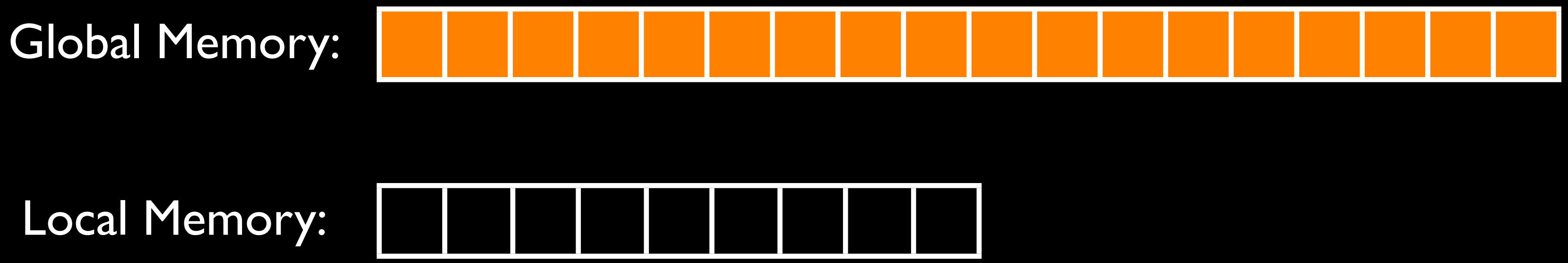
Before:



```
sum = 0;
for (i = 0; i < 100; ++i) {
    Copy A[i] to Local[0];
    sum += Local[0];
}
```

IBM Cell BE

Simple Array Tiling:



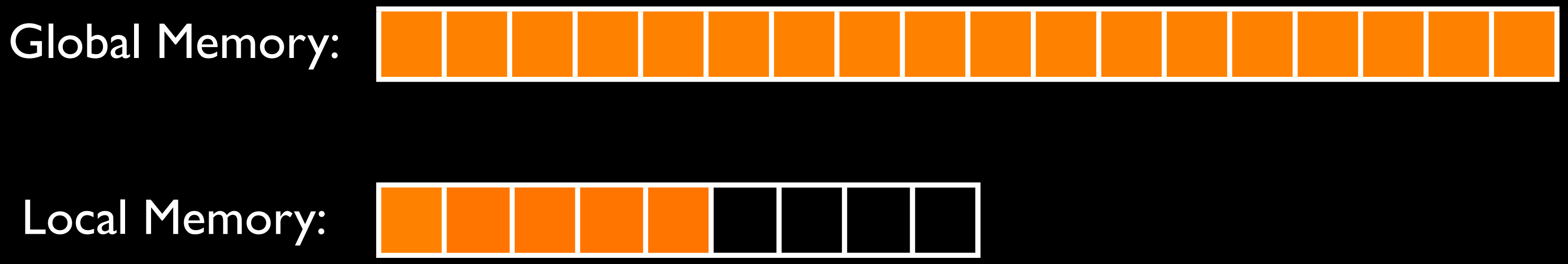
Global Memory:

Local Memory:

```
sum = 0;
for (j = 0; j < 100; j += 5) {
  Copy A[j:j+4] to Local[0:4]
  for (i = 0; i < 5; ++i) {
    sum += Local[i];
  }
}
```

IBM Cell BE

Simple Array Tiling:



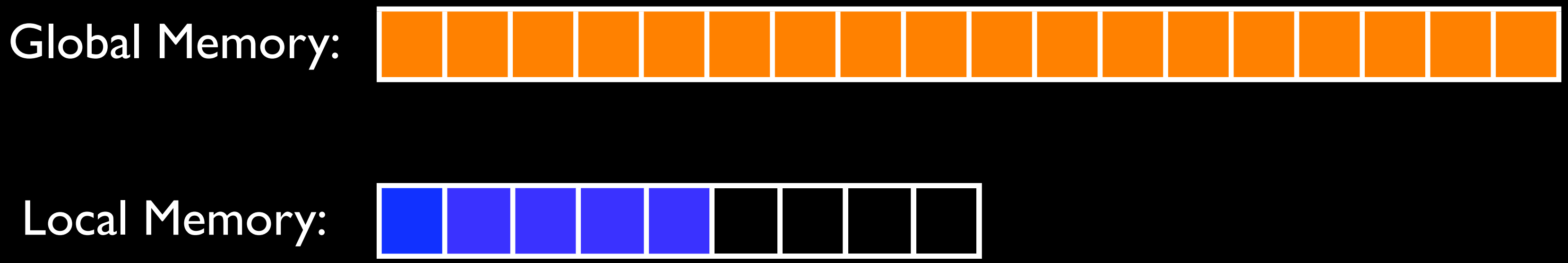
Global Memory:

Local Memory:

```
sum = 0;
for (j = 0; j < 100; j += 5) {
  Copy A[j:j+4] to Local[0:4]
  for (i = 0; i < 5; ++i) {
    sum += Local[i];
  }
}
```

IBM Cell BE

Simple Array Tiling:



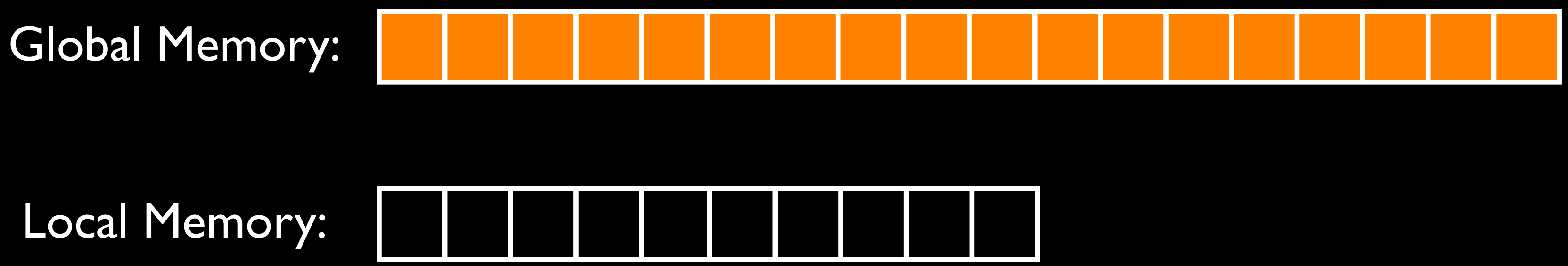
Global Memory:

Local Memory:

```
sum = 0;
for (j = 0; j < 100; j += 5) {
  Copy A[j:j+4] to Local[0:4]
  for (i = 0; i < 5; ++i) {
    sum += Local[i];
  }
}
```

IBM Cell BE

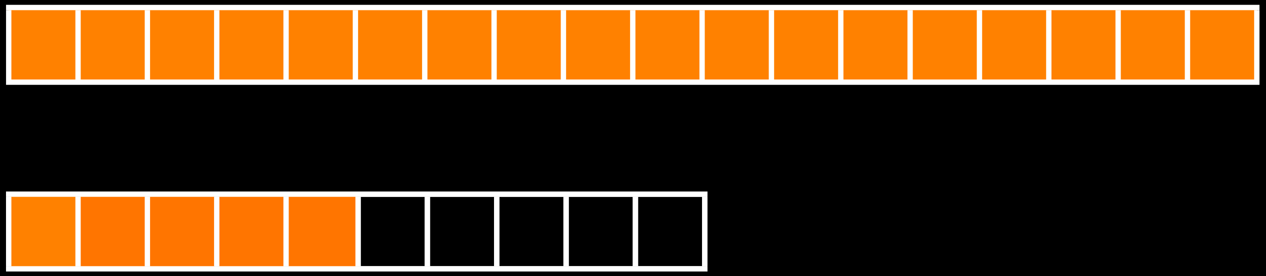
Double Buffering:



```
sum = 0;
current = Local;
next = Local + 5;
Copy A[0:4] to current[0:4];
for (j = 5; j < 100; j += 5) {
   StartCopy A[j:j+4] to next[0:4];
   for (i = 0; i < 5; ++i) sum += current[i];
   WaitCopy
   swap(next, current);
}
```

IBM Cell BE

Double Buffering:



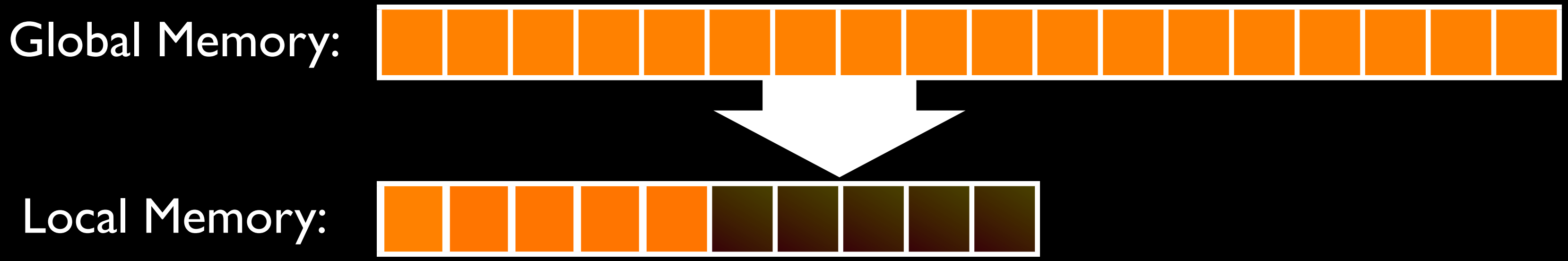
Global Memory:

Local Memory:

```
sum = 0;
current = Local;
next = Local + 5;
Copy A[0:4] to current[0:4];
for (j = 5; j < 100; j += 5) {
   StartCopy A[j:j+4] to next[0:4];
   for (i = 0; i < 5; ++i) sum += current[i];
   WaitCopy
   swap(next, current);
}
```

IBM Cell BE

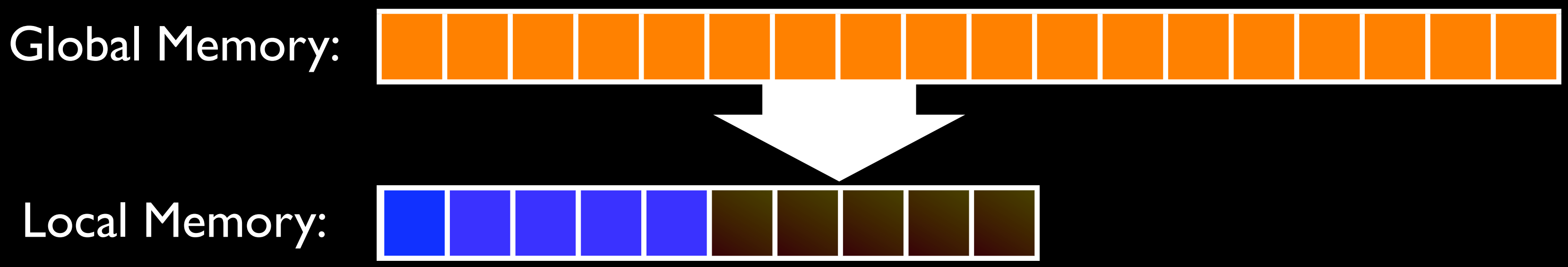
Double Buffering:



```
sum = 0;
current = Local;
next = Local + 5;
Copy A[0:4] to current[0:4];
for (j = 5; j < 100; j += 5) {
   StartCopy A[j:j+4] to next[0:4];
   for (i = 0; i < 5; ++i) sum += current[i];
   WaitCopy
   swap(next, current);
}
```

IBM Cell BE

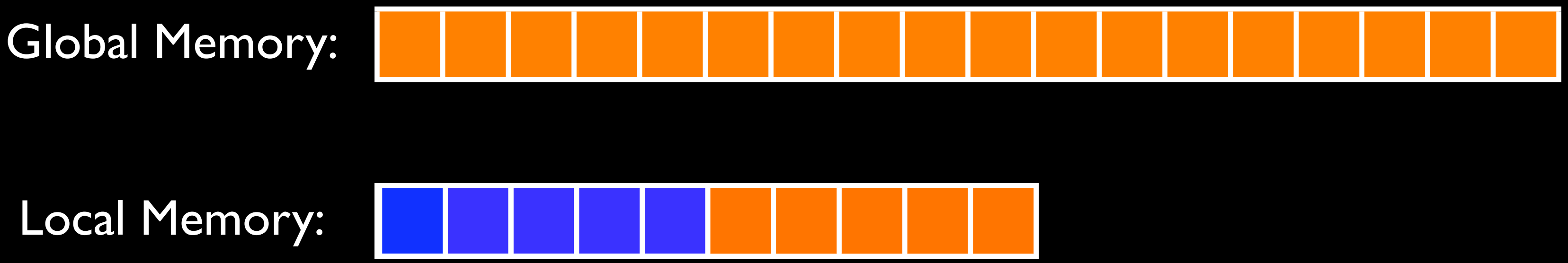
Double Buffering:



```
sum = 0;
current = Local;
next = Local + 5;
Copy A[0:4] to current[0:4];
for (j = 5; j < 100; j += 5) {
    StartCopy A[j:j+4] to next[0:4];
    for (i = 0; i < 5; ++i) sum += current[i];
    WaitCopy
    swap(next, current);
}
```

IBM Cell BE

Double Buffering:



Global Memory:

Local Memory:

```
sum = 0;
current = Local;
next = Local + 5;
Copy A[0:4] to current[0:4];
for (j = 5; j < 100; j += 5) {
   StartCopy A[j:j+4] to next[0:4];
   for (i = 0; i < 5; ++i) sum += current[i];
   WaitCopy
   swap(next, current);
}
```

IBM Cell BE

Triple Buffering

When?

An Observation

IBM Cell BE

An Observation

IBM Cell BE | NVIDIA G8x

An Observation

We had plenty to talk about over here...

...not so much over here.

Why?

IBM Cell BE | NVIDIA G8x

Jim’s Opinion:

General-Purpose Computation

Specialized Applications

Consider the Applications:

IBM Cell BE | NVIDIA G8x

Your Opinion?

IBM Cell BE | NVIDIA G8x