Dynamic Translation for EPIC Architectures

David R. Ditzel
Chief Architect for Hybrid Computing, VP IAG
Intel Corporation

Presentation for 8th Workshop on EPIC Architectures
April 24, 2010

Thesis: The future of computing belongs to EPIC Architectures

EPIC:

- Explicitly Parallel Instruction Computer

  or
- Exposed Parallelism Instruction Computer

- Parallelism exposed for software to exploit
- Examples – Itanium, GPGPU's, Transmeta Efficeon/Crusoe

My belief:

- EPIC is a more power efficient approach
- Dynamic translation will improve power advantages
- May be a different EPIC than we know today

Biggest challenge

Power is the limiter

We must move to more efficient computing structures or # cores could be limited

Simple Power Scaling Example

Power = Cdyn x Voltage^2 x Frequency + Leakage (33%)

Moore's Law says # devices can double every node

- 4 cores go to 128 cores over 10 years
- How does power limit this expectation?

With an upper power limit of ~100 Watts, how many cores?

Easy to calculate scaling per node:

- Voltage scaling about 0.9x
- Cdyn scaling about 0.8x
- Assume frequency increase of 1.2x

From this data we can see how many cores we can have if we do not change to a more efficient approach

Power Limits # of Big Cores

Year	2008	2010	2012	2014	2016	2018
Technology Node (nm)	45	32	22	15	11	8
Total Power	100					
Power/core	25					
Freq	3.0					
Voltage	1.0					
Cdyn/Core	5.6					
Expected #Cores	4	8	16	32	64	128
Power Limited #Cores	4					

Power Limits # of Big Cores

Year	2008	2010	2012	2014	2016	2018
Technology Node (nm)	45	32	22	15	11	8
Total Power	100	100	100	100	100	100
Power/core	25					
Freq	3.0	3.6	4.3	5.2	6.2	7.5
Voltage	1.0	0.9	0.8	0.7	0.7	0.6
Cdyn/Core	5.6	4.4	3.6	2.8	2.3	1.8
Expected #Cores	4	8	16	32	64	128
Power Limited #Cores	4					

Power Limits # of Big Cores

Year	2008	2010	2012	2014	2016	2018
Technology Node (nm)	45	32	22	15	11	8
Total Power	100	100	100	100	100	100
Power/core	25	19	15	12	9	7
Freq	3.0	3.6	4.3	5.2	6.2	7.5
Voltage	1.0	0.9	0.8	0.7	0.7	0.6
Cdyn/Core	5.6	4.4	3.6	2.8	2.3	1.8
Expected #Cores	4	8	16	32	64	128
Power Limited #Cores	4					

Power Limits # of Big Cores

Year	2008	2010	2012	2014	2016	2018
Technology Node (nm)	45	32	22	15	11	8
Total Power	100	100	100	100	100	100
Power/core	25	19	15	12	9	7
Freq	3.0	3.6	4.3	5.2	6.2	7.5
Voltage	1.0	0.9	0.8	0.7	0.7	0.6
Cdyn/Core	5.6	4.4	3.6	2.8	2.3	1.8
Expected #Cores	4	8	16	32	64	128
Power Limited #Cores	4	5	7	9	11	14

Power Limits # of Big Cores

Year	2008	2010	2012	2014	2016	2018
Technology Node (nm)	45	32	22	15	11	8
Total Power	100	100	100	100	100	100
Power/core	25	19	15	12	9	7
Freq	3.0	3.6	4.3	5.2	6.2	7.5
Voltage	1.0	0.9	0.8	0.7	0.7	0.6
Cdyn/Core	5.6	4.4	3.6	2.8	2.3	1.8
Expected #Cores	4	8	16	32	64	128
Power Limited #Cores	4	5	7	9	11	14

We need to improve the efficiency of each core or we will suffer severe performance reduction

So how do we build improved cores?

Premise

Change of perspective needed

- Software should be part of the picture
- Hardware co-designed with software increases the available options
- Software needs a simple model of the "cost" of an instruction
  - Out-of-order processors made this impossible
  - In-order EPIC processor can provide this simple model
- Software can do a very good job of scheduling, but only if the scheduling blocks are large enough
- Let's look at an example of how to increase block size and improve scheduling

Compiler optimization example

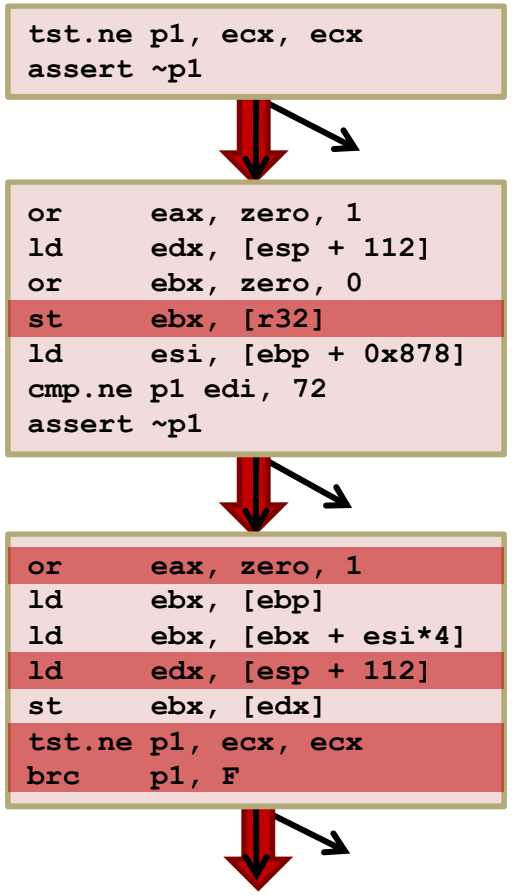
Conditional branches tend to have a very biased program behavior

- Exploitable by compiler

Correctness makes it difficult

- Fixup code for cold exits
- Exceptions

A little special purpose hardware can make it much easier



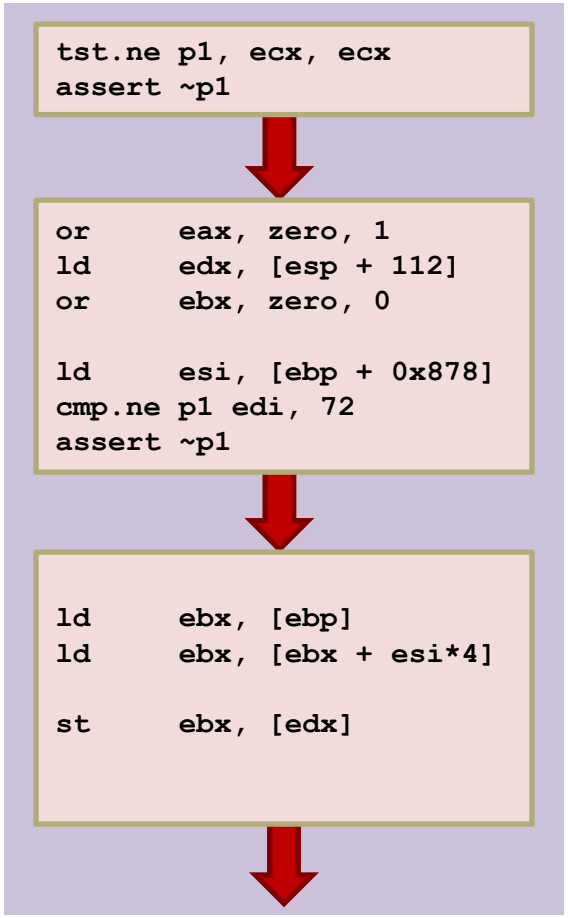
Hardware atomicity

Hardware executes a region of code
completely or not at all

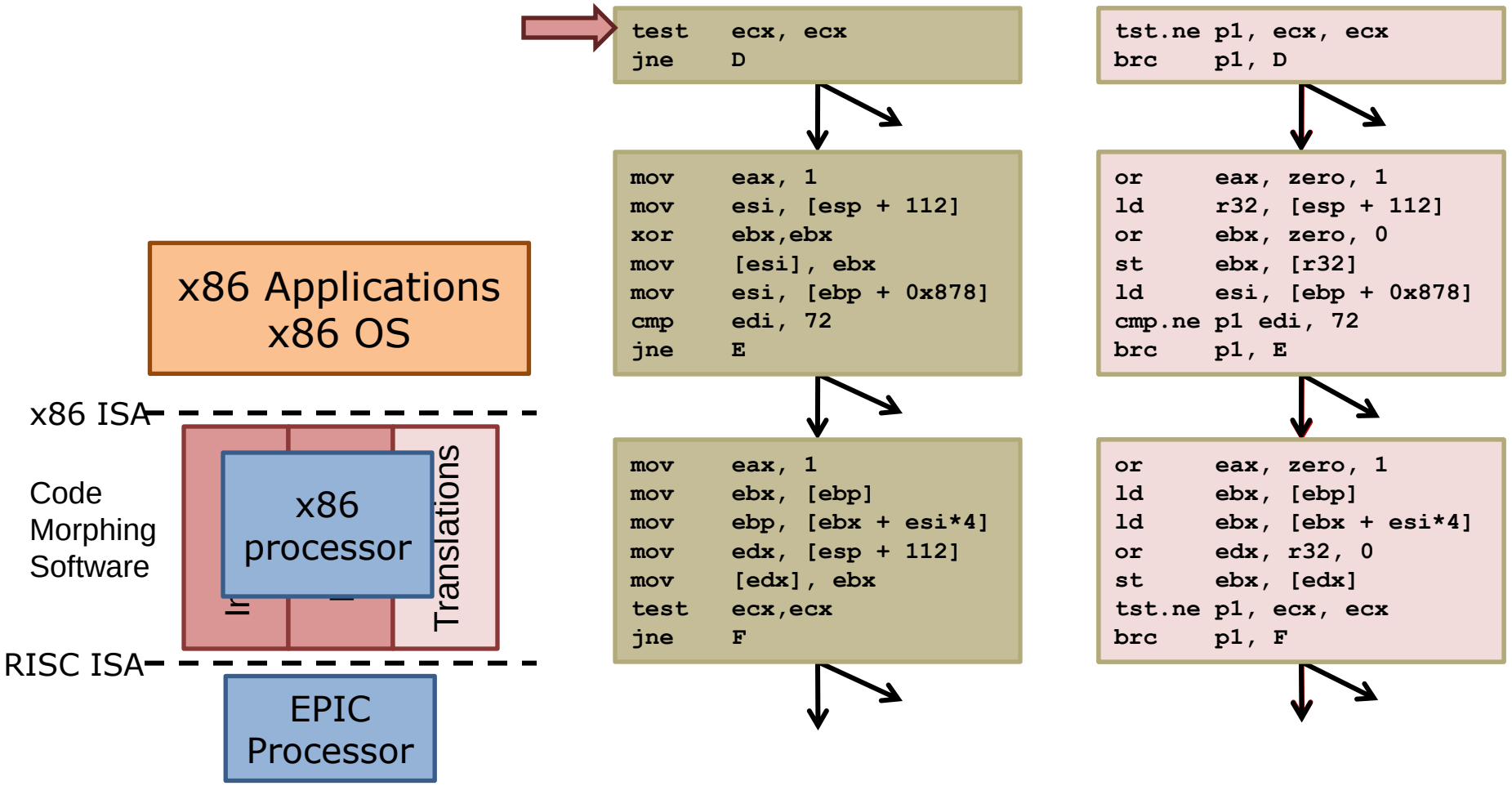
Common case is *fast*

Uncommon case rolls back

- Resume in non-specialized code



Dynamic binary translation



Efficeon Processor Example

Up to 6-issue/clock EPIC style architecture

- 2 loads or stores
- 2 integer ALU
- 2 SIMD
- 1 branch/call or other control

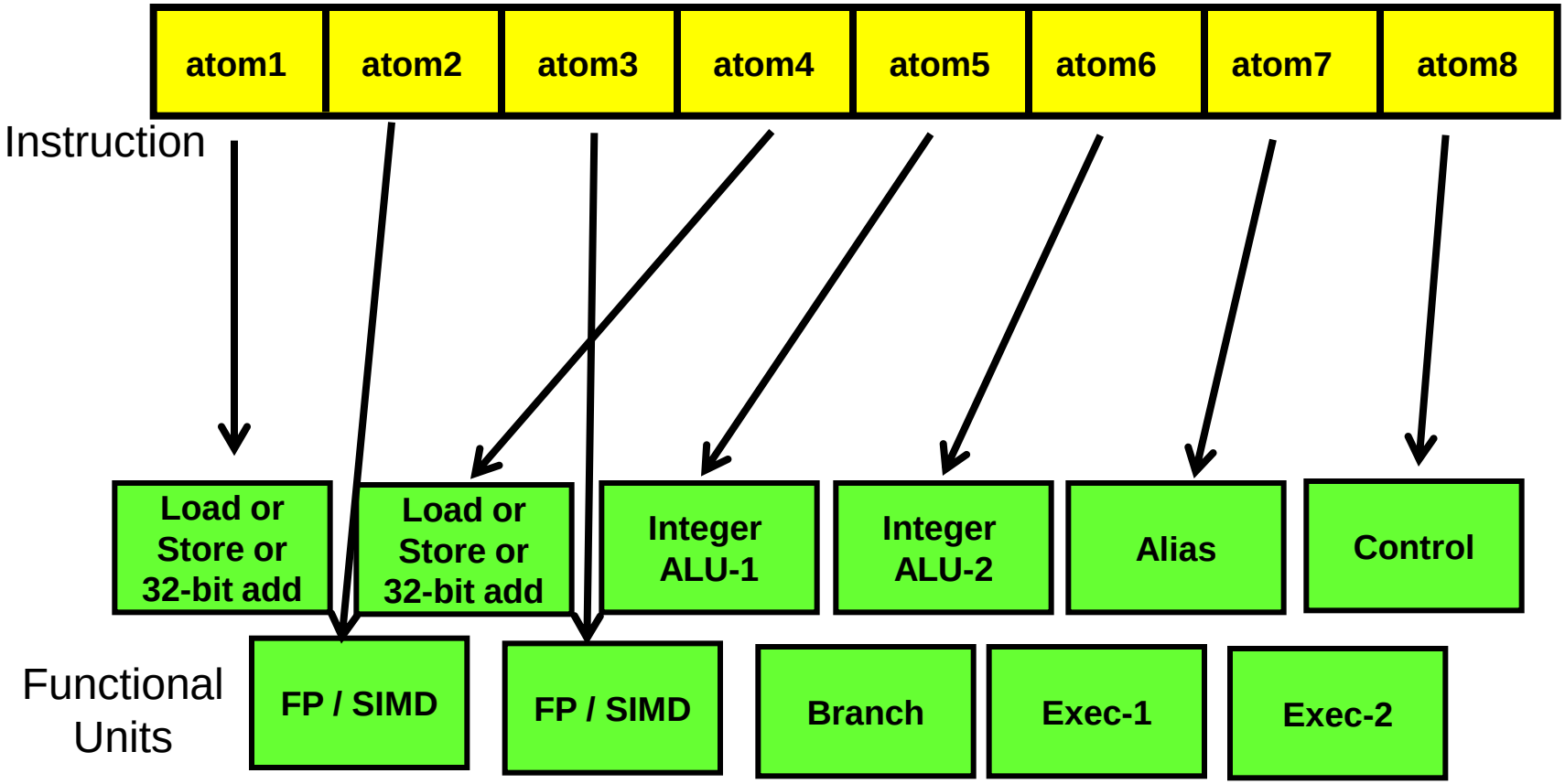
Co-designed with CMS

Includes hardware atomicity under software control

- Commit
- Rollback

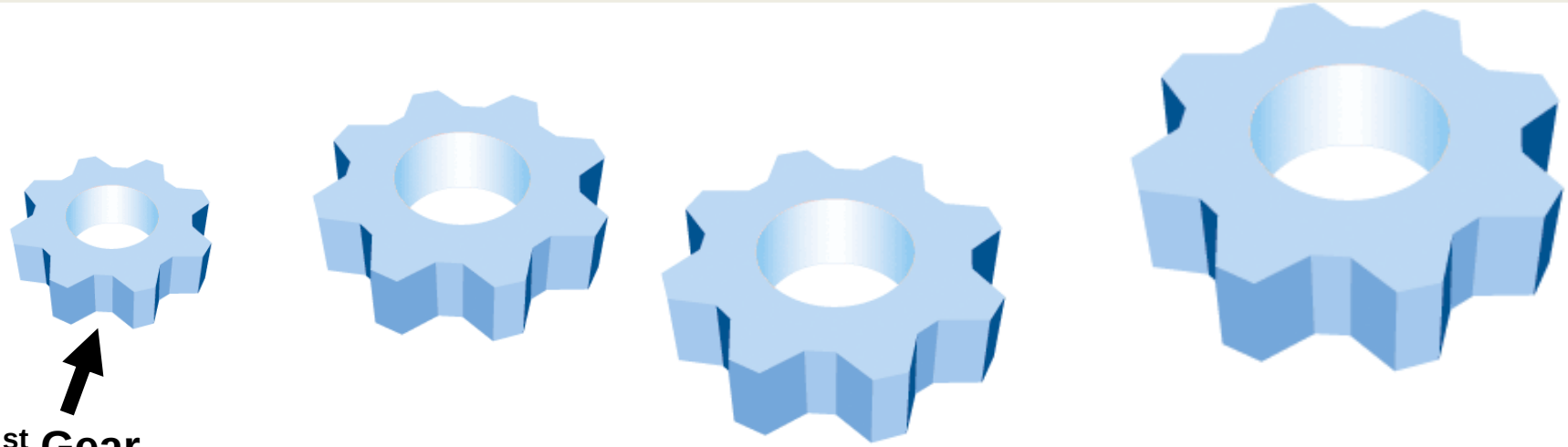
Efficeon Hardware Example

Each clock, processor can issue from one to six 32-bit instruction "atoms" to 11 functional units



Code Morphing Software

4 Gear System Significantly Improved Responsiveness and Overall Performance



1st Gear
Executes 1 instruction at a time

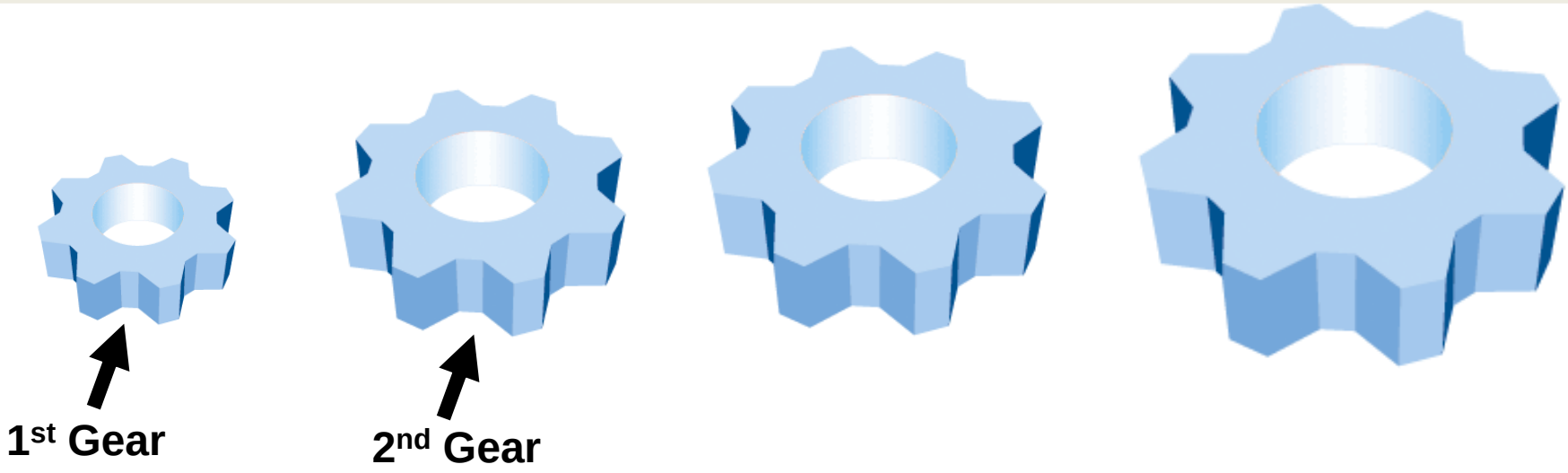
- Profiles code at runtime
- Gathers data for flow analysis
- Gathers branch frequencies and directions
- Detects load/store typing (IO vs memory)

Filters out infrequently executed code

No startup cost
Lowest speed

Code Morphing Software

4 Gear System Significantly Improved Responsiveness and Overall Performance



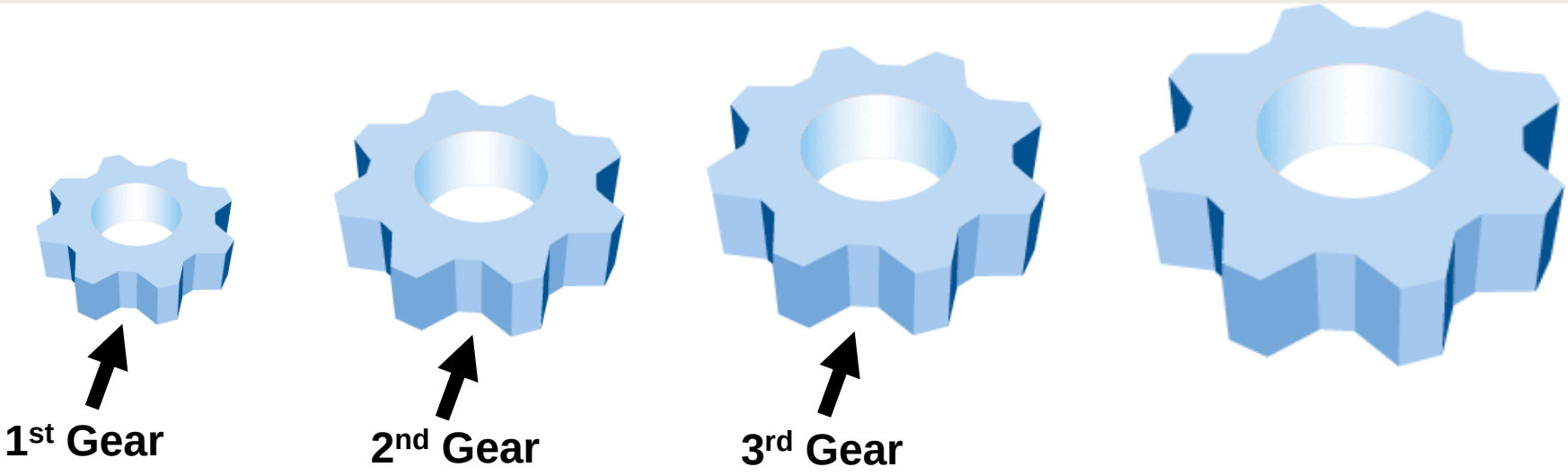
Uses profile data to create initial translations after code reaches 1st threshold.

- Translates a “Region” of up to100 x86 instructions.
- Adds flow graph “Shape” information
- Light Optimization
- “Greedy” scheduling

Low translation overhead
Fast execution

Code Morphing Software

4 Gear System Significantly Improved Responsiveness and Overall Performance



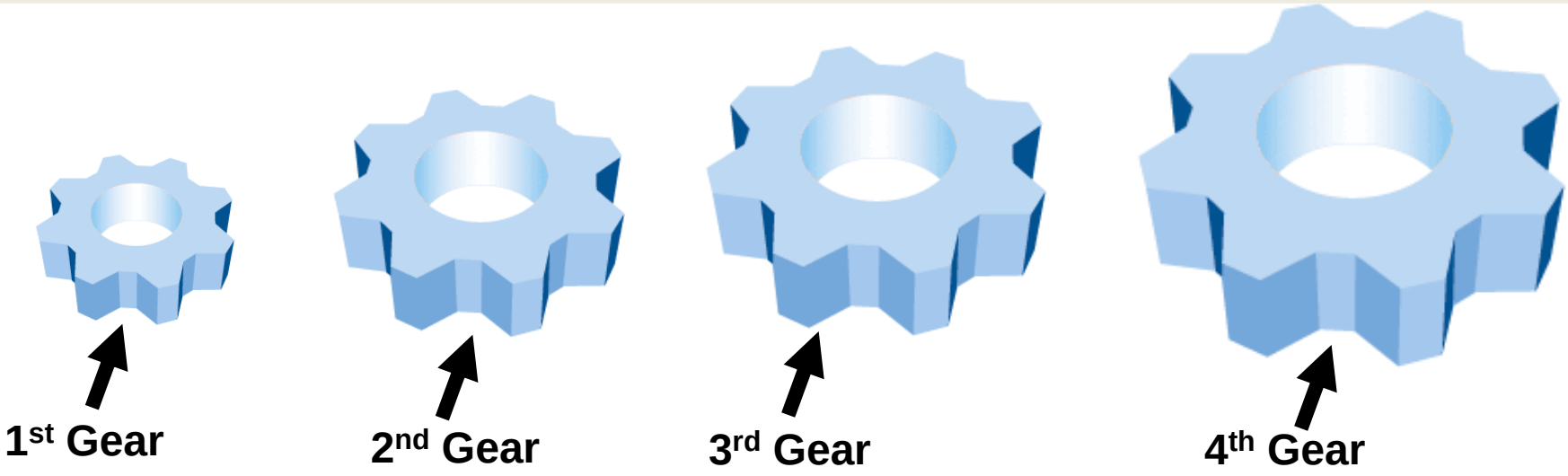
Further optimizes the 2nd gear regions

- Common sub-expression elimination
- Memory re-ordering
- Significant code optimization
- Critical path scheduling

Medium translation overhead
Faster execution

Code Morphing Software

4 Gear System Significantly Improved Responsiveness and Overall Performance



Most advanced optimizations
for “hottest” code regions.

- Splices together multiple regions
- Optimizes across region boundaries
- Used advanced behavioral data
- Critical path scheduling

Highest translation overhead
Fastest execution

CMS Translator Optimization

Sophisticated translation optimizer

- *Quickly* applies many optimizations
  - if-conversion, loop unrolling, constant folding and propagation, common-subexpression elimination, dead-code-elimination, loop-invariant code motion, superblock scheduling

- New optimizations must have low overhead

Example from Vortex

Dynamic opportunities in CMS translation

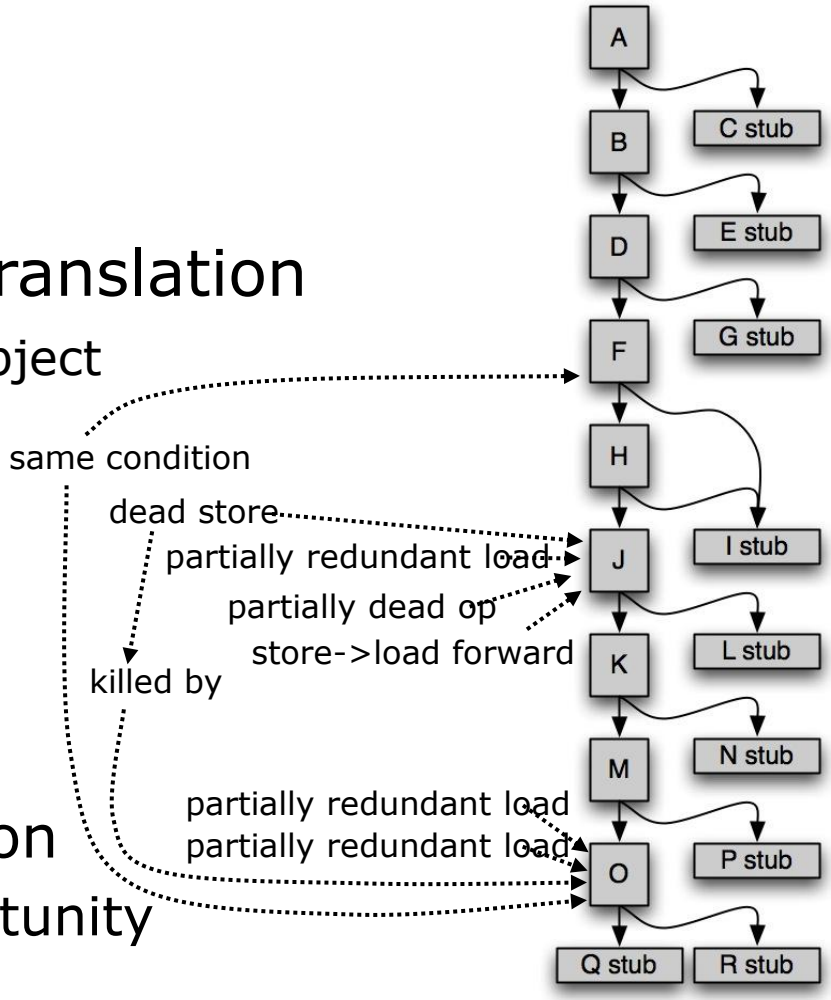
- Hottest method in Vortex: OaGetObject

Potential to eliminate x86 insts

- 56 -> 47 dynamic x86 insts
- 19% reduction

CMS relies on superblock abstraction

- Does not expose available opportunity



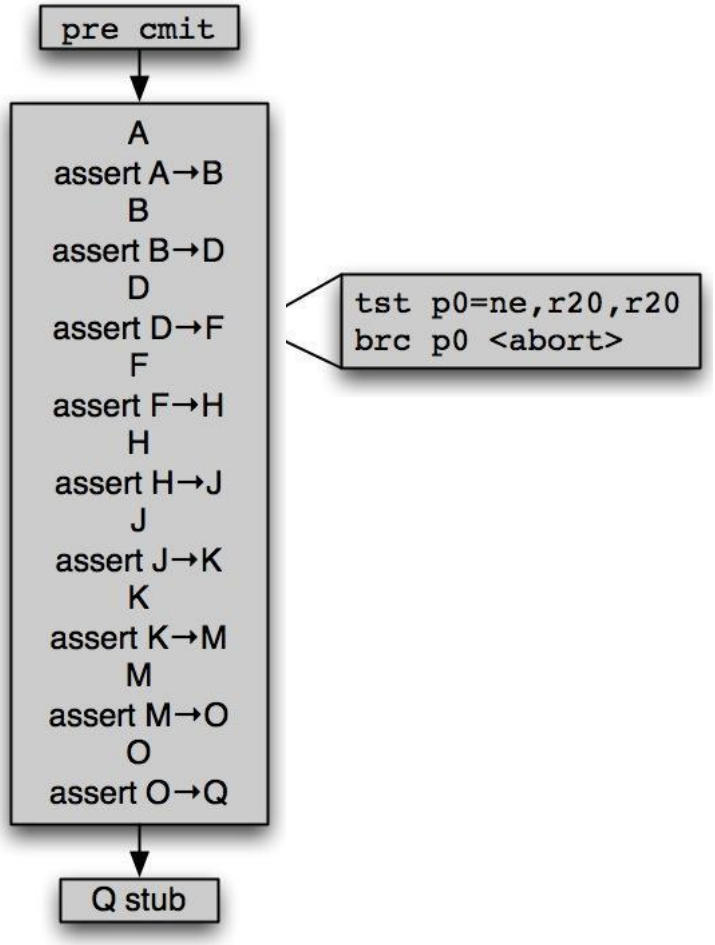
Atomic region abstraction

Atomic regions trivially expose opportunity

Convert biased control flow into assert operations

- Represent as dataflow op in IR
- Traditional optimizations can now exploit speculative opportunity
- Emit as conditional branch to jump to rollback and recover

Retry in the interpreter or another translation



Atomic region benefits

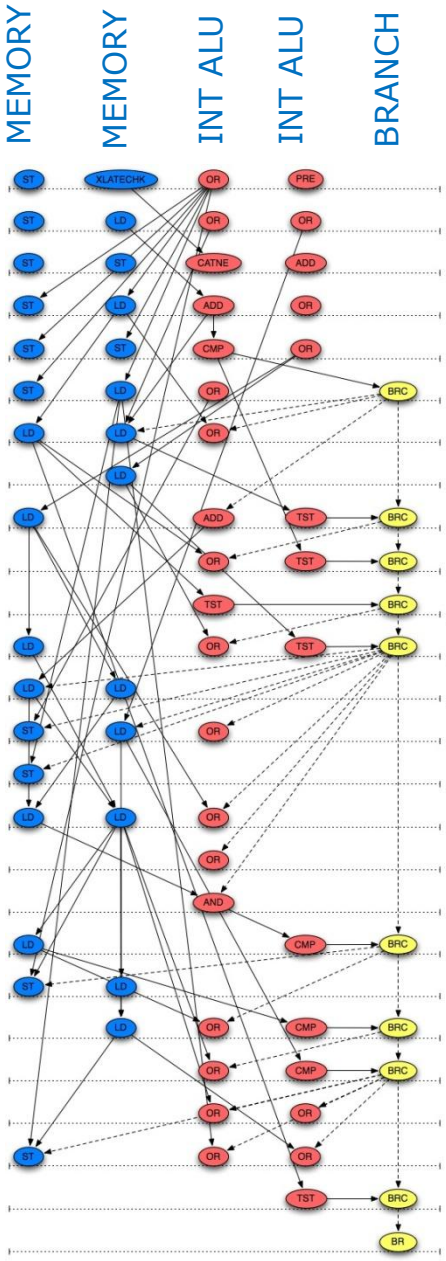
Dependence graph shown

Atomic region trivially enables optimizer to eliminate operations

- 88 -> 73 Efficeon operations
- 17% reduction

Relaxes scheduling constraints

- 26 -> 19 cycles
- 27% reduction



Why we need EPIC architectures

EPIC architectures offer many advantages

Simplified hardware
- Simpler to design
- Smaller cores means more cores per die

Enables software scheduling
- EPIC architectures are easier for DBT to schedule
- Better scheduling is the key to future performance gains

Power
- In-order pipelines for EPIC are power efficient
- Less hardware for OOO means lower power
- More amenable to new power saving techniques

Why we need Dynamic Translation

Good reasons for Dynamic Binary Translation (DBT)

Innovation

- To allow processor innovation not tied to particular instruction sets
- Using DBT to provide backwards compatibility
- DBT system hidden from standard software – CMS as microcode

Performance

- To enable new means to improve processor performance
- DBT can provide access to new performance features

Power

- Dynamic optimization is good for power
- e.g., optimizing away half the instructions is twice as energy efficient

Conclusions

Binary Translation with EPIC architectures are a good combination.

Special purpose hardware support is needed, co-designed with software, in order to provide good performance and power efficiency.

Special care is needed to keep translation overhead low.

Many opportunities for clever hardware/software co-design tradeoffs

This is a technological approach still in its infancy

Prediction: Dynamic Binary Translation will become a basic technique used in future processor design, as integral as logic gates and microcode are today.

END OF SLIDES