New Challenges in Microarchitecture and Compiler Design

Fred Pollack

Intel Fellow

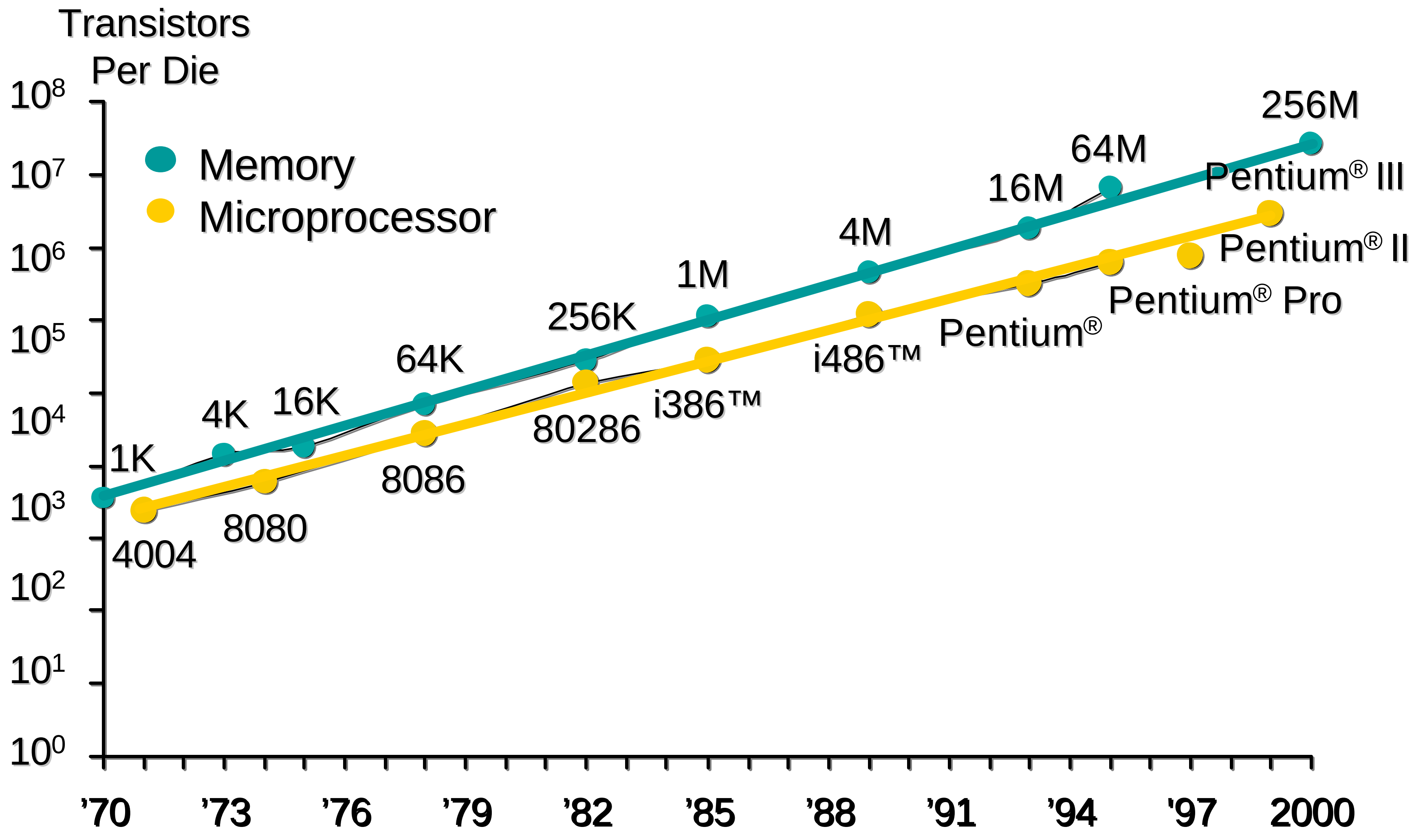
Director of Microprocessor Research Labs

Intel Corporation

fred.pollack@intel.com

Contributors:
Jesse Fang
Tin-Fook Ngai

Moore's Law



Source: Intel

Moore's Law

- **The number of transistors on a chip will double every generation (18-24 months)**
  - **2,300 transistors on the 4004 in 1971**
  - **About 120 million transistors on the Pentium® III Xeon processor in 2000**
  - **An increase of 50000x in 29 years**

Performance Doubles Every 18 Months

100X Increase in last 10 years

- Intel 486 processor at 33Mhz in 1990
- To the Pentium® 4 processor at 1.5 GHz

Sources of Performance (approximate)

- 20X from Process and Circuit Technology
- 4X from Architecture
- 1.4X from Compiler Technology

Architecture Performance

Microarchitecture

- **Deeper Pipelining to increase frequency**
- **Execution of more instructions in parallel**
- **On die Caches**

System Architecture

- **Buses, e.g. 33Mhz 486 bus to 400Mhz Pentium® 4 bus**
  - 132Mbytes/sec vs. 3.2Gbytes/sec
  - On die caches grew from 4Kbytes to as much as 2Mbytes on the Pentium III Xeon™ processor
- **Memory Bandwidth**
  - 66 Mbytes/sec in a 486 system
  - 3.2 Gbytes/sec in a Pentium 4 system
- **IO Bandwidth**
  - 3 Mbytes/sec on an ISA bus
  - 1 Gbyte/sec on AGP4X

In the Last 25 Years Life was Easy

Doubling of transistor density every 30 months

Increasing die sizes, allowed by

- **Increasing Wafer Size**
- **Process technology moving from "black art" to "manufacturing science"**

Þ ***Doubling of transistors every 18 months***

And, only constrained by cost & mfg limits

But how efficiently did we use the transistors?

Performance Efficiency of µarchitectures

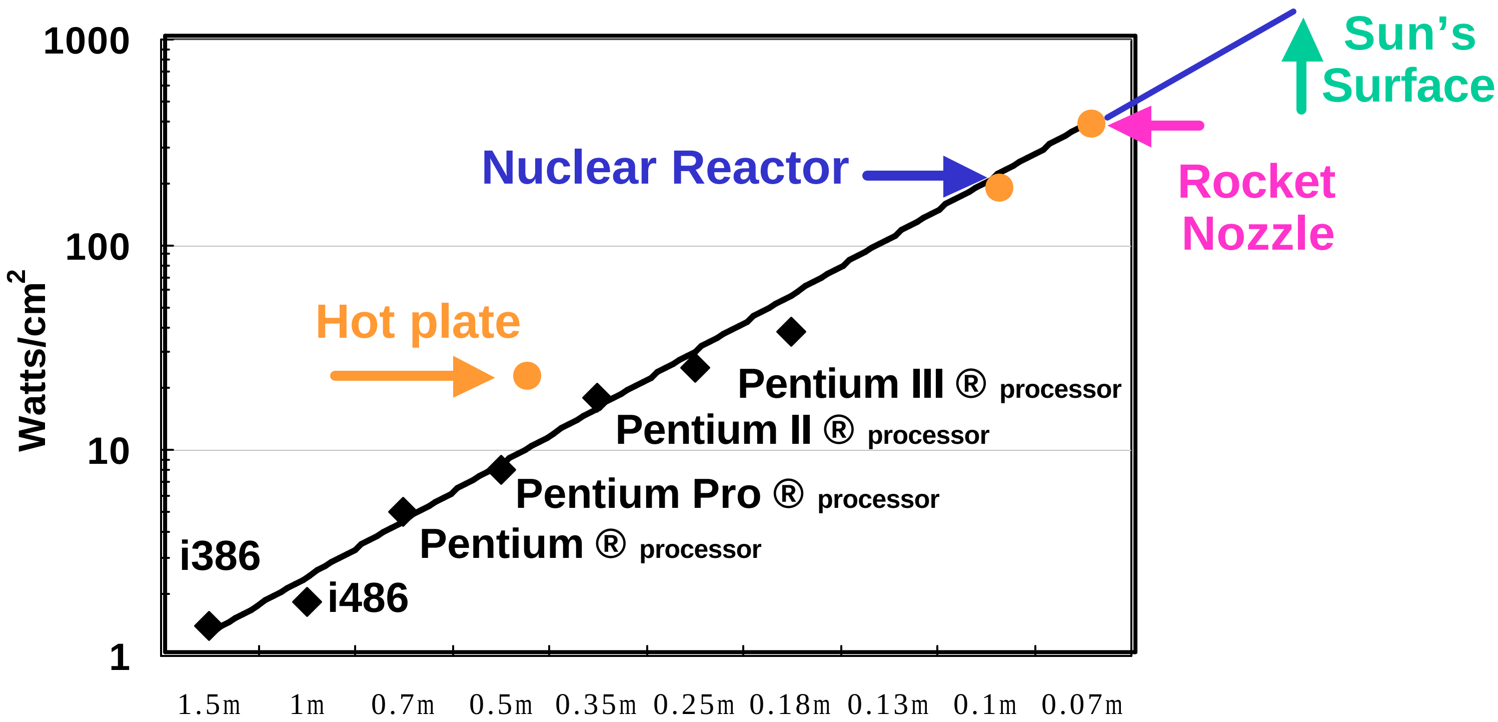
Tech	Old mArch	mm (linear)	New mArch	mm (linear)	Area
1.0m	i386C	6.5	i486	11.5	3.1
0.7m	i486C	9.5	Pentium® proc	17	3.2
0.5m	Pentium® proc	12.2	Pentium Pro proc	17.3	2.1
0.18m	Pentium III proc	10.3	Pentium 4 proc	?	2+

Implications: (in the same technology)

1. **New mArch ~ 2-3X die area of the last mArch**
2. **Provides 1.4-1.7X integer performance of the last mArch**

We are on the Wrong Side of a Square Law

Power density continues to get worse



Surpassed hot-plate power density in 0.5m

Not too long to reach nuclear reactor

If we continue on the current trend, which we can't

Implications

- We can’t build microprocessors with ever increasing die sizes
- The constraint is power – not manufacturability
- Must use transistors efficiently and target for valued performance

Microarchitecture Directions

- **Logic Transistor growth constrained by power – not mfg**
  - **At constant power, 50% per process generation vs. over 200% in past**
- **Current Directions in microarchitecture that help**
  - **SIMD ISA extensions**
  - **On-die L2 caches**
  - **Multiple CPU cores on die**
  - **Multithreaded CPU**
- **Key Challenges for future Microarchitectures**
  - **Special purpose performance**
  - **Increase execution efficiency: improved prediction and *confidence***
  - **Break the data-flow barrier, but in a power efficient manner**

Changing Landscape of Computing

Server applications

- Higher throughput demands
- Multithread apps on multiprocessor

Internet applications

- Independent user tasks/threads
- Across internet on different platforms
- Security

Peer-to-peer applications

Pervasive Computing

Move from Machine-based to human-based interfaces

New Computing Environment

New languages & language support

- Java, C#, XML
- Runtime environment: Virtual Machine
- Portability for data and code

New challenges in the environment

- Driven by run-time program/data
- Multithreaded program execution

SW Application Trends

Beyond performance
- Maintainability, Reliability, Availability, Scalability
- Ease of Use

Shorter time-to-market

New software development techniques
- Object oriented
- Software reuse (component based)

Performance without excessive tuning/profiling

Opportunity

Break the dataflow barrier by increasing the cooperation between the compiler and microarchitecture in the execution of the new computing models

New Challenges

Beyond ILP: thread level parallelism

- Thread level parallelism
- Speculative multithreaded microarchitecture

Dynamic compilation and optimization

- Dynamic compilation of ILs for new languages

New Challenges

Beyond ILP: thread level parallelism

- Thread level parallelism
- Speculative multithreaded microarchitecture

Dynamic compilation and optimization

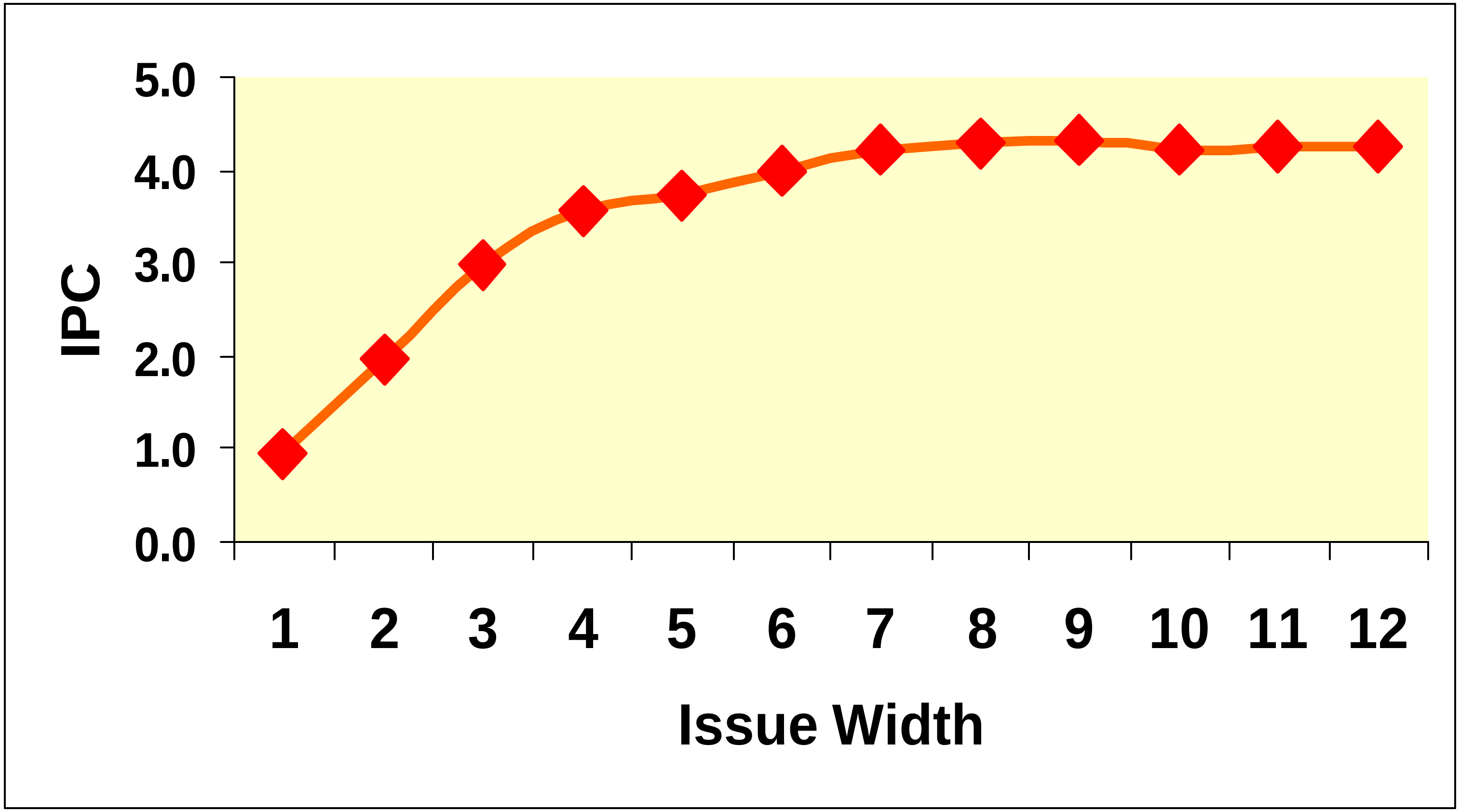
- Dynamic compilation of ILs for new languages

Thread Level Parallelism

- **High throughput for multithreaded applications**
  - **Not only for servers but also clients**
- **How to boost single thread application performance on multithreaded microarchitecture**
  - **Speculation: from instruction level to block level**
  - **Prediction: from branch (control flow) to value (data flow)**
  - **Locality: from instruction/data to computation**
  - **New optimization techniques to best exploit the above**

Static ILP is hitting its limit

In-order scheduling microarchitecture with perfect memory



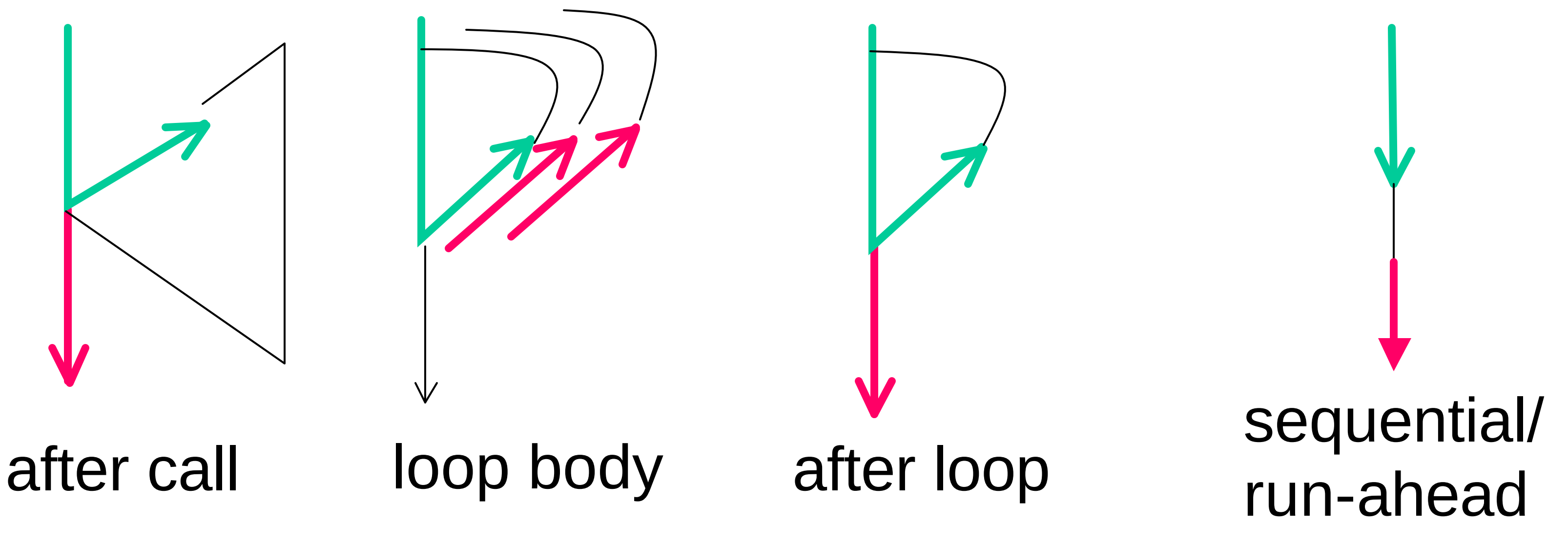
Benchmark GCC: Issue Width vs IPC

From Intel Microprocessor Research Labs

Speculative Multithreading

An application program is decomposed into multiple threads

- Call and after-call threads
- Loop iteration threads
- Main and run-ahead threads



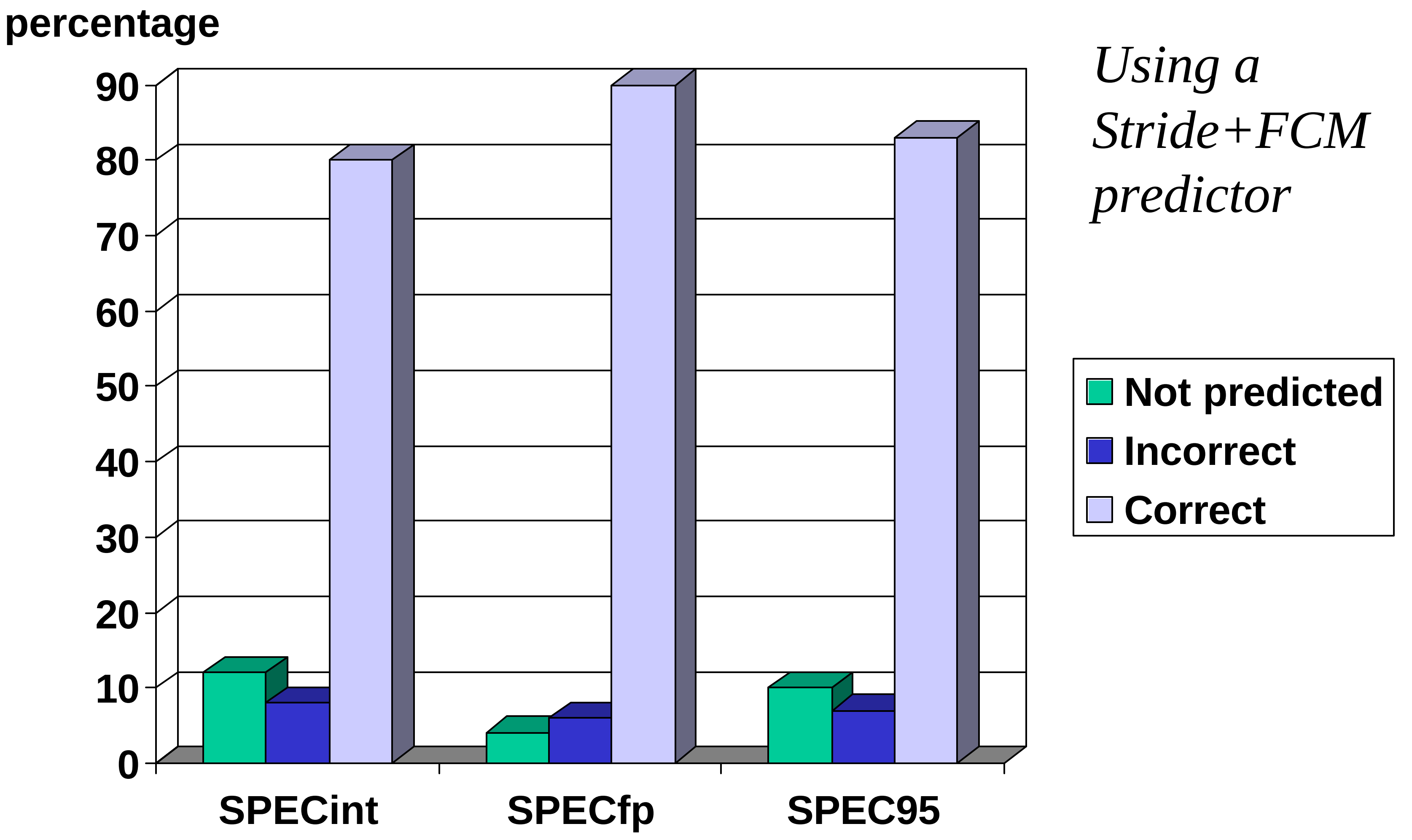
Speculative Multithreading

- **With microarchitectural support, the threads are speculatively executed in parallel**
  - **If correct, increased parallelism**
  - **Otherwise (hopefully only occasionally), squash the speculative threads and re-execute**
    - ***Re-execution is faster due to data prefetching and early branch resolution by the speculative execution***

Thread-Level Parallelism with Value Prediction

- Some real data dependences between threads can be removed by correctly predicting the data value
  - Loop induction variables
  - Procedure return values
  - Variables with almost constant runtime values
- Common value predictors
  - Last value predictors
  - Stride predictors
  - Finite context methods (FCM)

Value Prediction Accuracy



Using a Stride+FCM predictor

From ***[Rychlik/Faistl/Krug/Shen98]***

An Example

A difficult-to-parallelize loop in decompress() from compress:

```
while (1) {
    code = getcode();                      // get next code
    if ( code == -1 ) goto exit;           // exit if no more code
    if (…) {                               //  if special marker, reset and get next code
        free_ent = … ;
        code = getcode();
        if ( code == -1 )  goto exit; }    // exit if no more code
    incode = code;
    if ( code >= free_ent ) {              // for a special case, process the last code
        *stackp++ = … ;
        code = oldcode;  }
    while ( code >= 256 )  {               // lookup code sequence and push onto stack
        *stackp++ = … ; …; }
    do ... while ( stackp > base );        // pop and output code sequence from stack
     if (…)  free_ent++ ;                  // if a new code, generate a new table entry
     oldcode = incode;  }                  // update the last code
```

Are these real dependences?

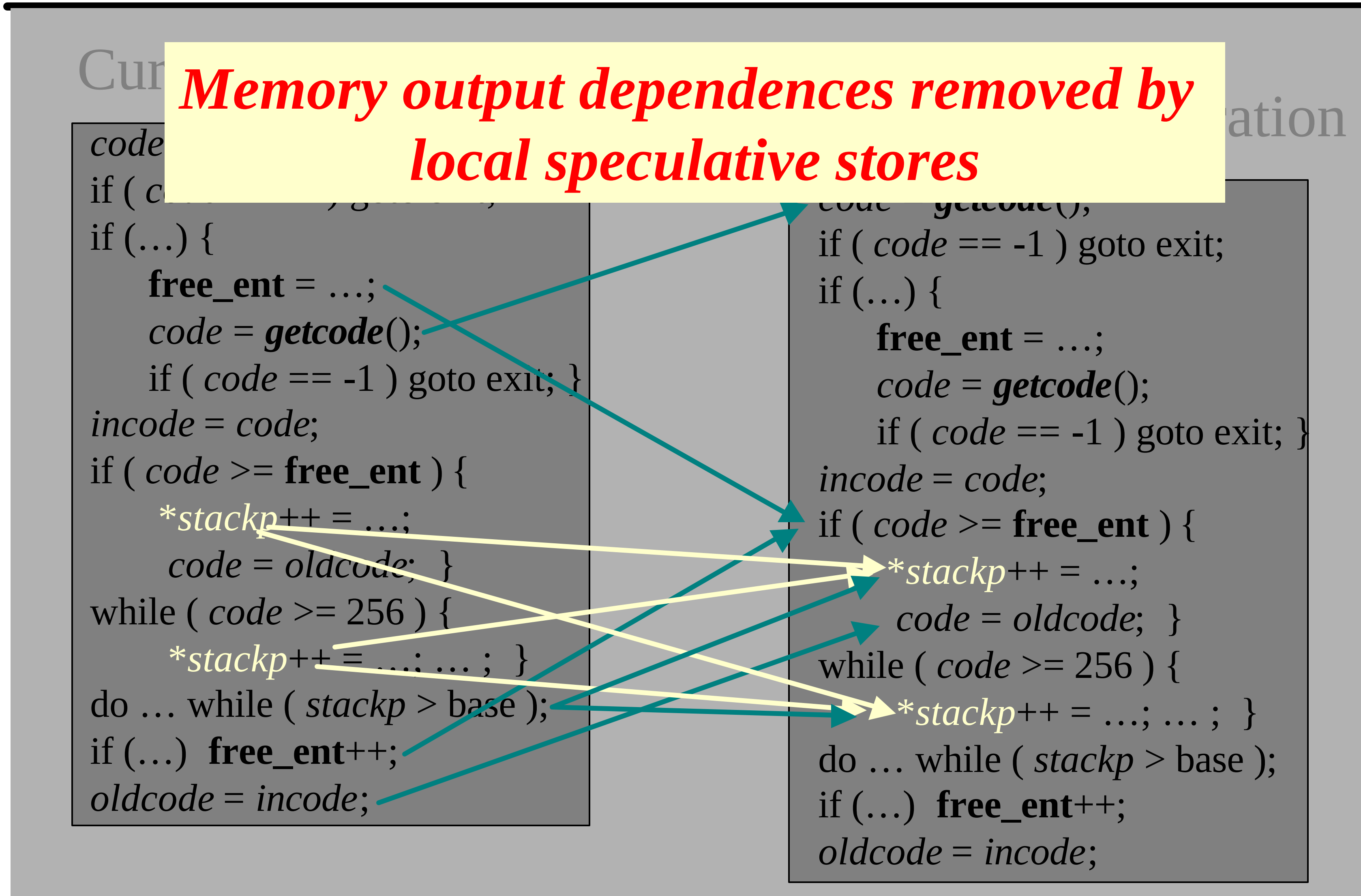
Current iteration

```
code = getcode();
if ( code == -1 ) goto exit;
if (…) {
    free_ent = …;
    code = getcode();
    if ( code == -1 ) goto exit; }
incode = code;
if ( code >= free_ent ) {
     *stackp++ = …;
      code = oldcode;  }
while ( code >= 256 ) {
      *stackp++ = …; … ;  }
do … while ( stackp > base );
if (…)  free_ent++;
oldcode = incode;
```

Speculative next iteration

```
code = getcode();
if ( code == -1 ) goto exit;
if (…) {
    free_ent = …;
    code = getcode();
    if ( code == -1 ) goto exit; }
incode = code;
if ( code >= free_ent ) {
     *stackp++ = …;
      code = oldcode;  }
while ( code >= 256 ) {
      *stackp++ = …; … ;  }
do … while ( stackp > base );
if (…)  free_ent++;
oldcode = incode;
```

No real output dependences



Current iteration

```
code = getcode();
if ( code == -1 ) goto exit;
if (…) {
     free_ent = …;
     code = getcode();
     if ( code == -1 ) goto exit; }
incode = code;
if ( code >= free_ent ) {
      *stackp++ = …;
       code = oldcode;  }
while ( code >= 256 ) {
       *stackp++ = …; … ;  }
do … while ( stackp > base );
if (…)  free_ent++;
oldcode = incode;
```

Speculative next iteration

```
code = getcode();
if ( code == -1 ) goto exit;
if (…) {
     free_ent = …;
     code = getcode();
     if ( code == -1 ) goto exit; }
incode = code;
if ( code >= free_ent ) {
      *stackp++ = …;
       code = oldcode;  }
while ( code >= 256 ) {
       *stackp++ = …; … ;  }
do … while ( stackp > base );
if (…)  free_ent++;
oldcode = incode;
```

Effective value prediction

Current iterat

```
code = getcode();
if ( code == -1 ) go
if (…) {
    free_ent = …;
    code = getcode();
    if ( code == -1 ) goto exit; }
incode = code;
if ( code >= free_ent ) {
     *stackp++ = …;
     code = oldcode;  }
while ( code >= 256 ) {
     *stackp++ = …; … ;  }
do … while ( stackp > base );
if (…)  free_ent++;
oldcode = incode;
```

Often false

Often true

Each new iteration begins with

- **free_ent** ***increment by one (mostly)***
- ***the same stackp value (always)***

```
if (…) {
    free_ent = …;
    code = getcode();
    if ( code == -1 ) goto exit; }
incode = code;
if ( code >= free_ent ) {
     *stackp++ = …;
     code = oldcode;  }
while ( code >= 256 ) {
     *stackp++ = …; … ;  }
do … while ( stackp > base );
if (…)  free_ent++;
oldcode = incode;
```

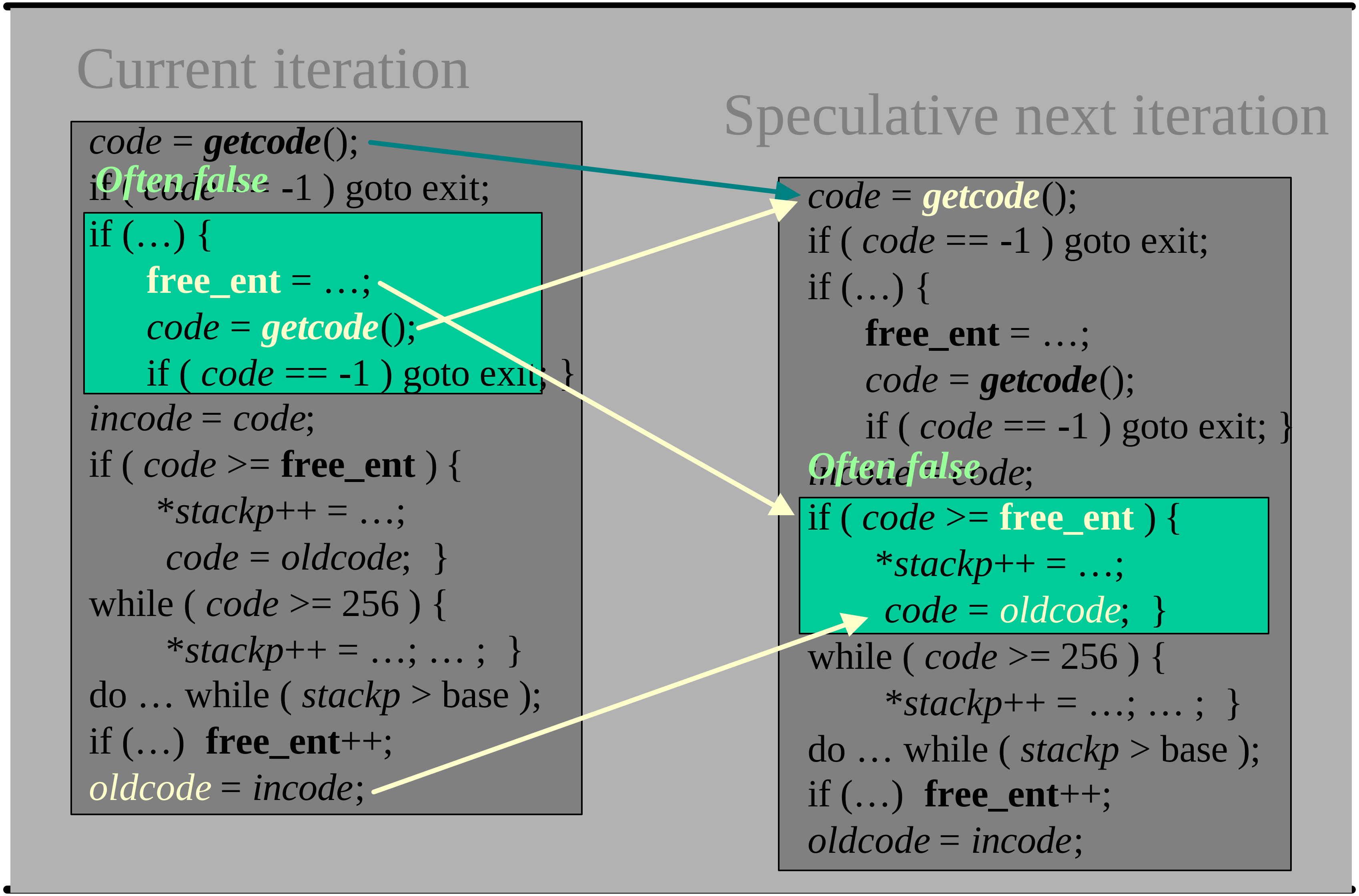
Current iteration

```
code = getcode();
if ( code == -1 ) goto exit;
if (…) {
     free_ent = …;
     code = getcode();
     if ( code == -1 ) goto exit; }
incode = code;
if ( code >= free_ent ) {
       *stackp++ = …;
        code = oldcode;  }
while ( code >= 256 ) {
        *stackp++ = …; … ;  }
do … while ( stackp > base );
if (…)  free_ent++;
oldcode = incode;
```

Speculative next iteration

```
code = getcode();
if ( code == -1 ) goto exit;
if (…) {
     free_ent = …;
     code = getcode();
     if ( code == -1 ) goto exit; }
incode = code;
if ( code >= free_ent ) {
       *stackp++ = …;
        code = oldcode;  }
while ( code >= 256 ) {
        *stackp++ = …; … ;  }
do … while ( stackp > base );
if (…)  free_ent++;
oldcode = incode;
```

False dependences during runtime



Runtime parallel MT execution

Current iteration

```
code = getcode();
if ( code == -1 ) goto exit;
if (…) {
     free_ent = …;
     code = getcode();
     if ( code == -1 ) goto exit; }
incode = code;
if ( code >= free_ent ) {
      *stackp++ = …;
       code = oldcode;  }
while ( code >= 256 ) {
       *stackp++ = …; … ;  }
do … while ( stackp > base );
if (…)  free_ent++;
oldcode = incode;
```

Speculative next iteration

```
code = getcode();
if ( code == -1 ) goto exit;
if (…) {
     free_ent = …;
     code = getcode();
     if ( code == -1 ) goto exit; }
incode = code;
if ( code >= free_ent ) {
      *stackp++ = …;
       code = oldcode;  }
while ( code >= 256 ) {
       *stackp++ = …; … ;  }
do … while ( stackp > base );
if (…)  free_ent++;
oldcode = incode;
```

New Challenges

Beyond ILP: thread level parallelism

- Thread level parallelism
- Speculative multithreaded microarchitecture

Dynamic compilation and optimization

- Dynamic compilation of ILs for new languages

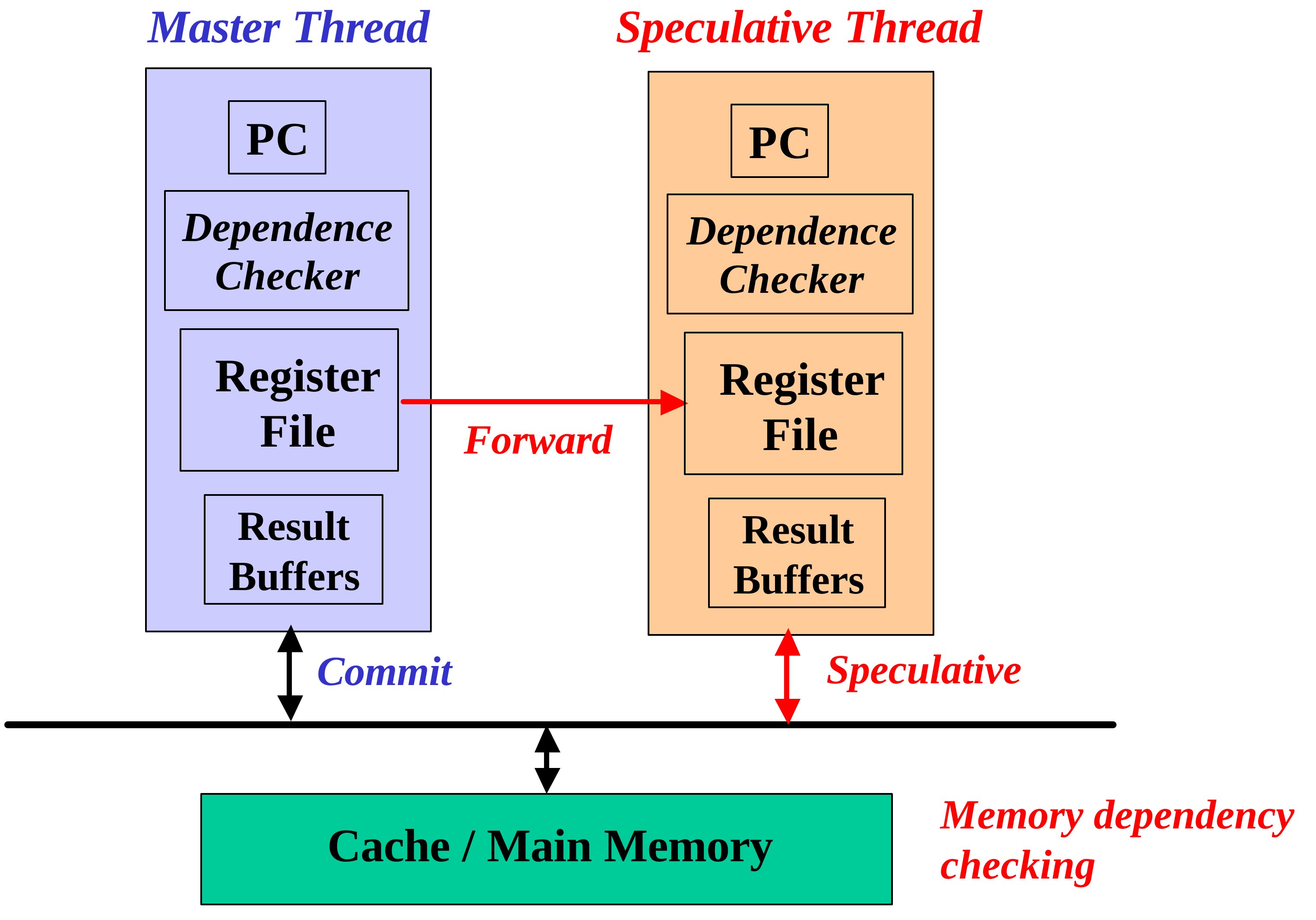
Multithreaded Microarchitectures

- Dedicated local context per running thread
- Efficient resource sharing
  - Time sharing
  - Space sharing
- Fast thread synchronization/communication
  - Explicit instructions
  - Implicit via shared registers/cache/buffer

Speculation Support

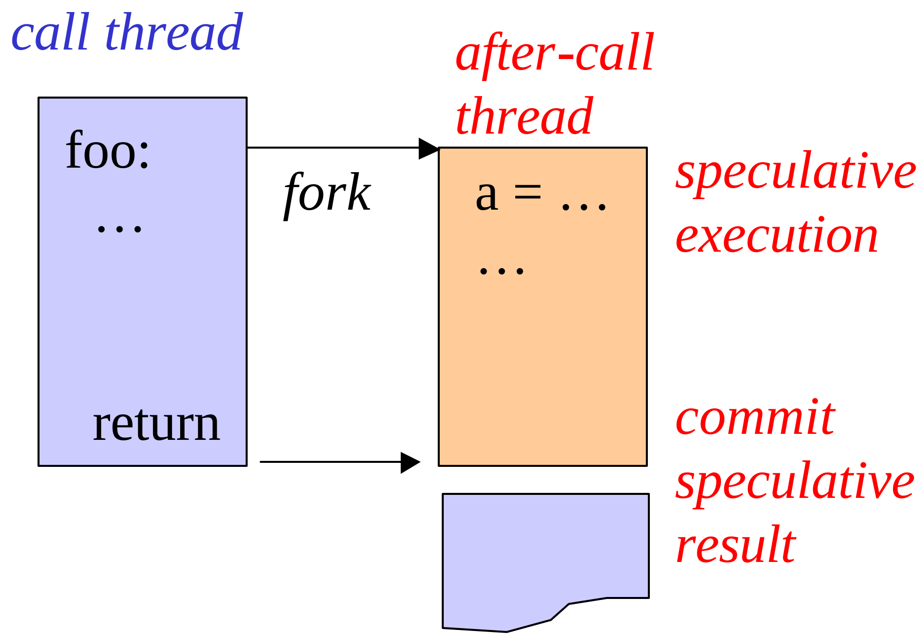
- Checkpointing
- Runtime dependence checking
  - Register data dependence
  - Memory store-load dependence
- Recovery if misspeculated
  - Squash speculative threads and re-execute
- Committing speculative results

Speculative MT microarchitecture



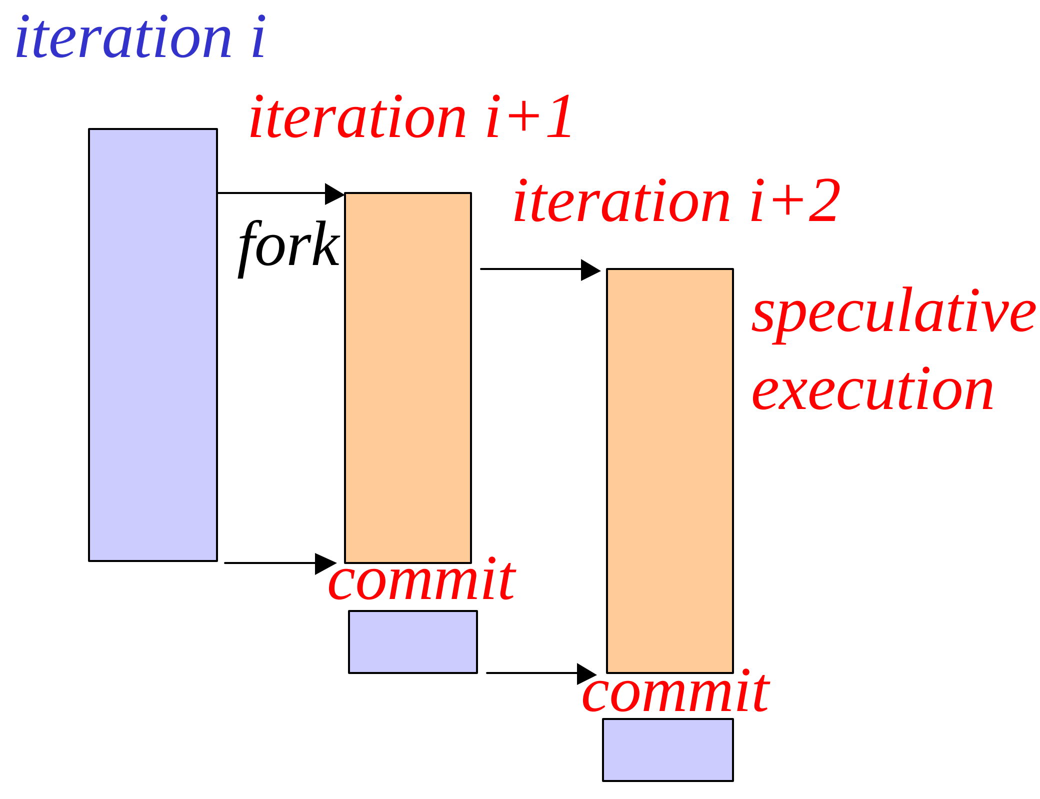
Procedure Speculation

r = foo( );
a = …;
…

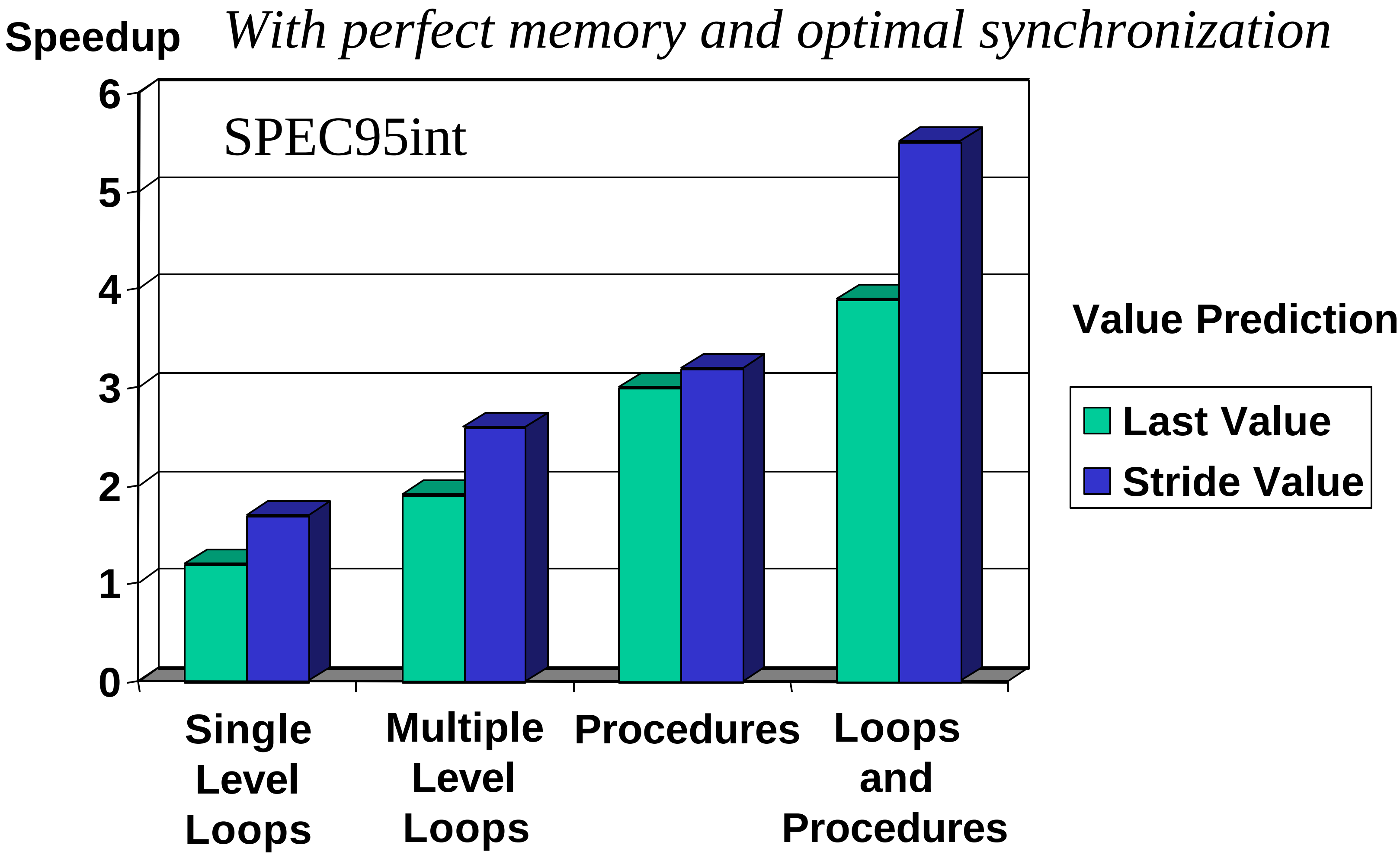


Loop Speculation

```
for (i = 0; i < n; i++)
{
      …
}
```



Performance Potential in SpecMT



From ***[Oplinger/Heine/Lam99]***

Challenges in Speculative MT

- **Speculative thread computation model**
- **Speculative MT microarchitecture support**
  - **Extremely low cost thread communication/synchronization**
  - **Fast and effective checkpointing**
  - **Fast recovery and commit: To squash or to selectively reexecute**
  - **Speculative thread scheduling and throttling**
  - **Load balancing**
  - **Cache/memory subsystem support**
- **Speculative MT compilation**
  - **Dependence analysis for speculative threads**
  - **Identification of the most opportunistic threads**
  - **Code optimization to minimize misspeculations**

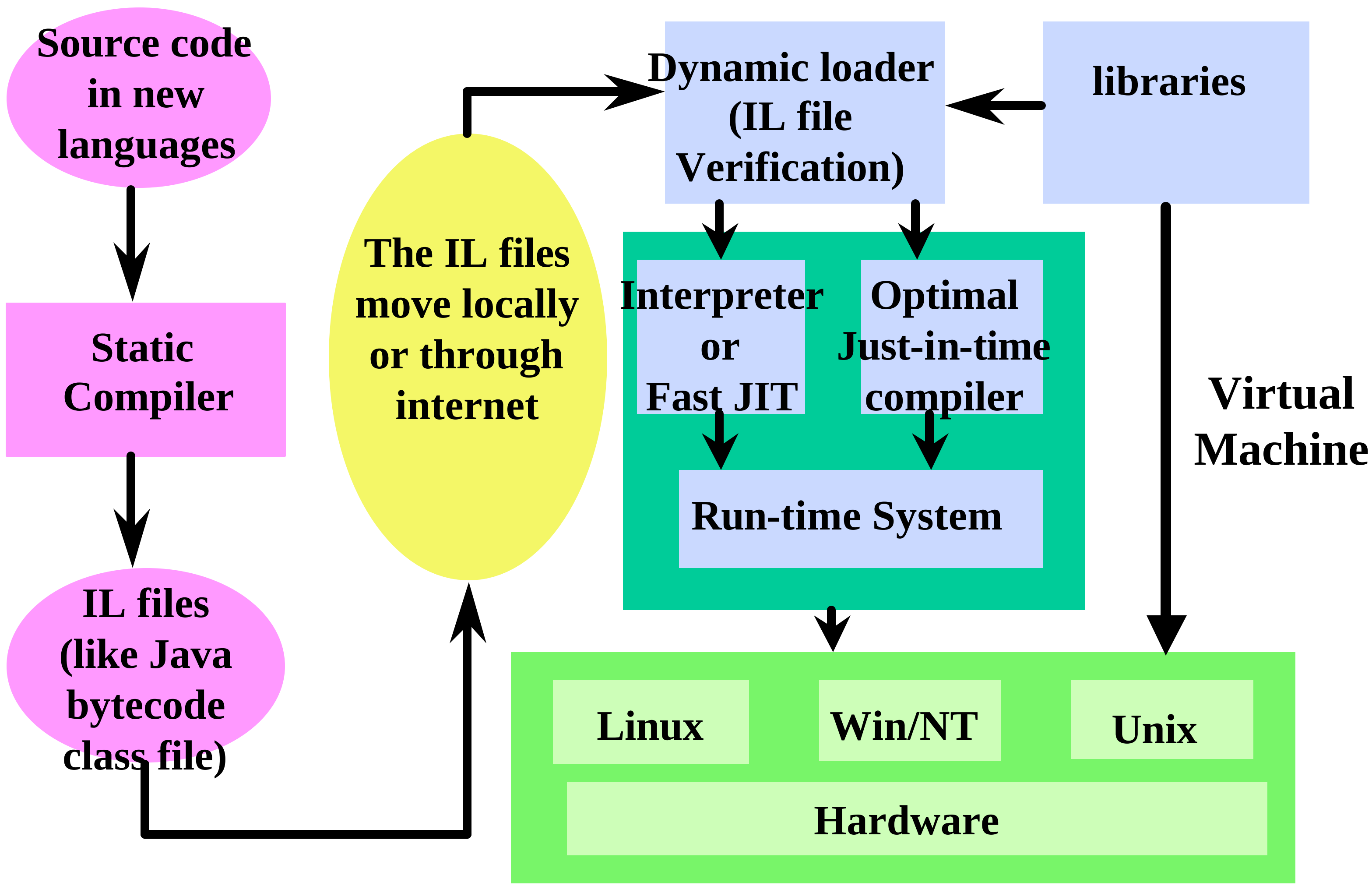
New Challenges

- **Beyond ILP: thread level parallelism**
  - **Thread level parallelism**
  - **Speculative multithreaded microarchitecture**
- **Dynamic compilation and optimization**
  - **Dynamic compilation of ILs for new languages**

Dynamic Compilation

- **Dynamic compilation of ILs for new languages such as bytecode for Java and MSIL for C#**
- **Runtime platform includes**
  - **Runtime environment: Virtual Machine**
  - **Dynamic memory management: Garbage collection**
  - **Dynamic loading and unloading**
  - **Dynamic optimization: Just-In-Time compiler**
  - **Security: Runtime security check**

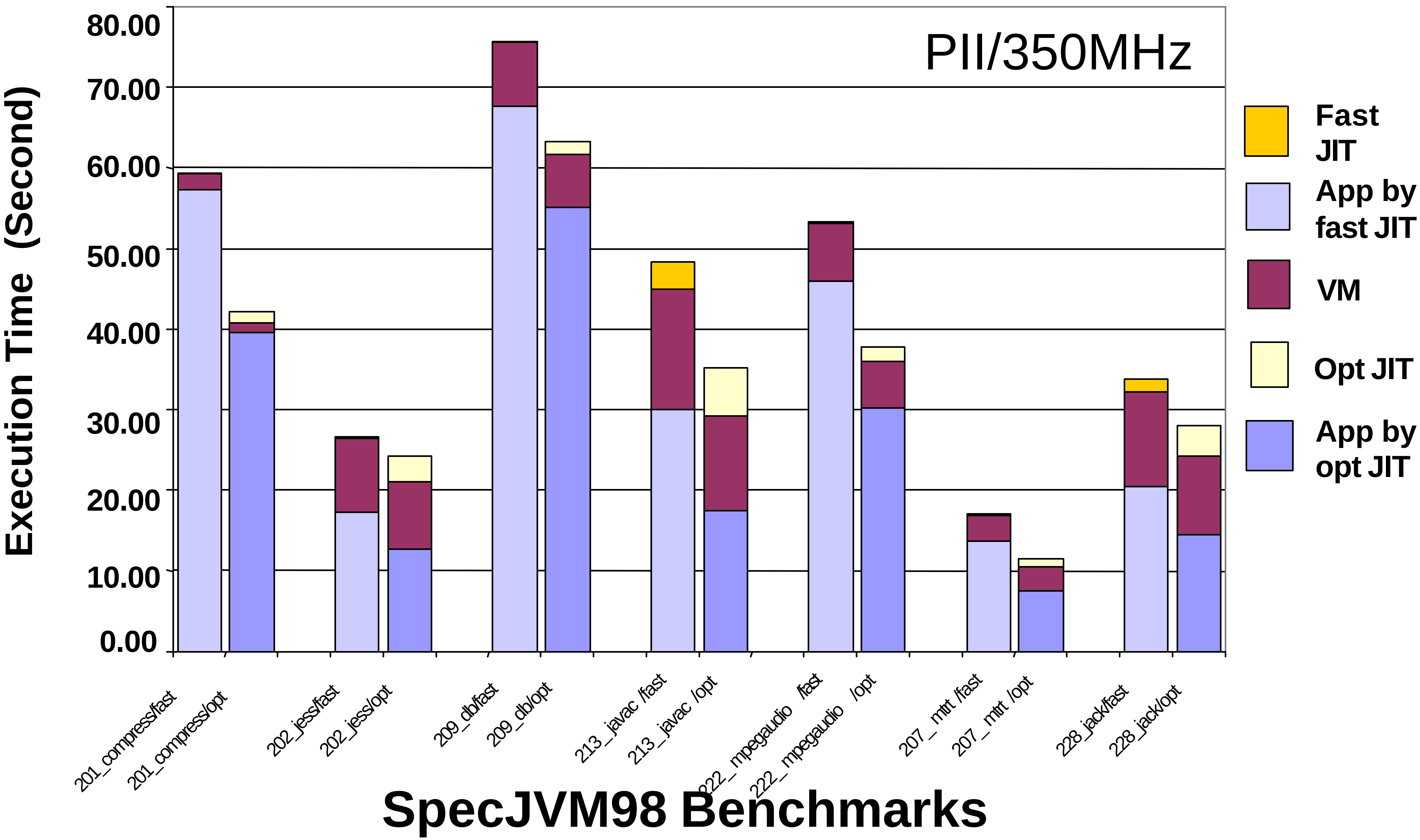
Dynamic Compilation Environment



Dynamic Compilation Strategies

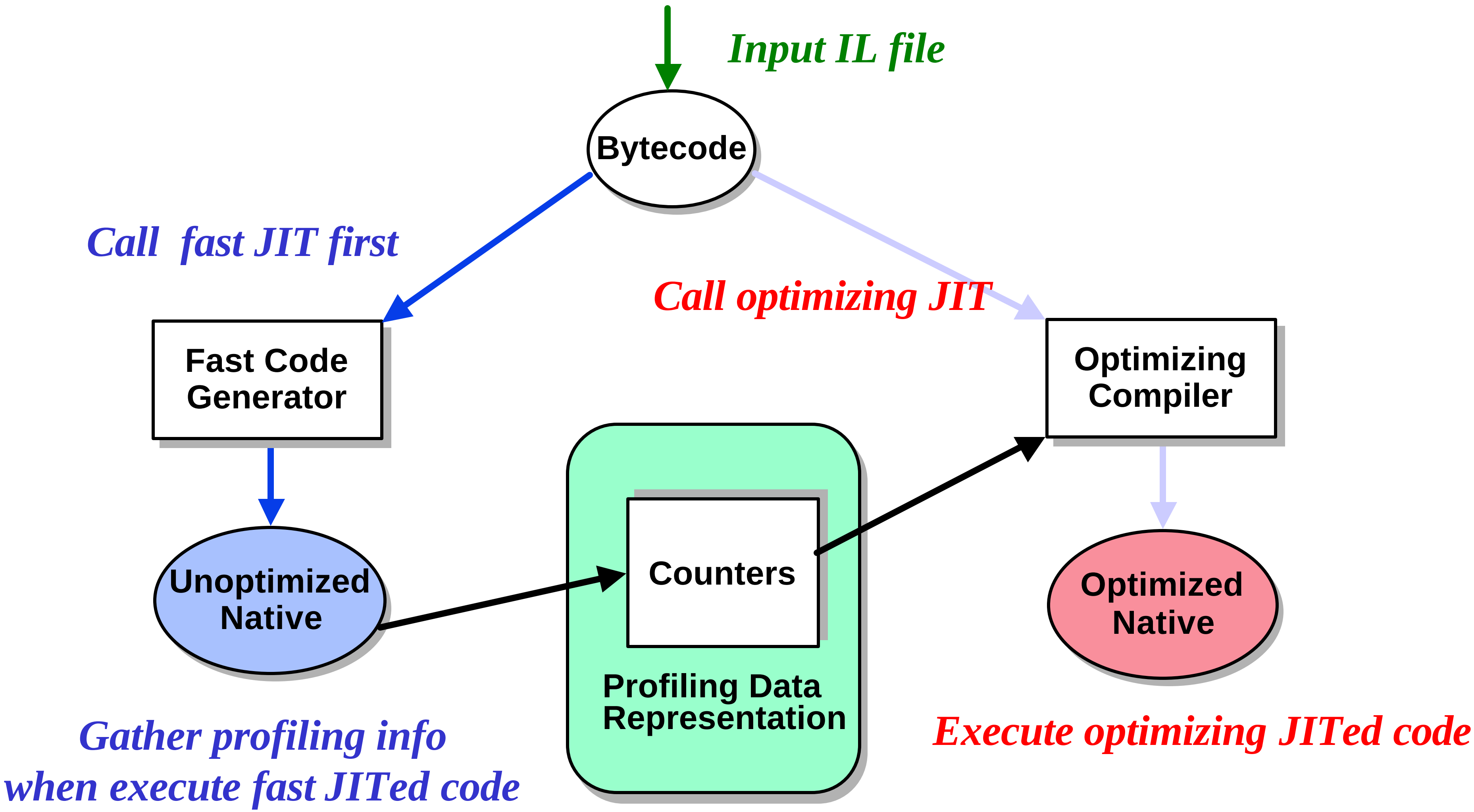
- **Lightweight optimization**
  - **Fast compilation time**
  - **Reasonable good performance**
- **Heavyweight optimization**
  - **Slow compilation time**
  - **Better performance**
- **90-10 rule**
  - **90% methods: lightweight**
  - **10% methods: heavyweight**
- **Tradeoff: Compilation time vs code execution time**

JIT Overhead and Performance



Source: Intel/MRL JIT 1999

JIT Compiler Infrastructure



Challenges of JIT Compiler

- **Efficient exception handling**
- **Bounds checking elimination**
- **Efficient synchronization**
- **Efficient support for garbage collection**
- **Effective use of profiling information**
  - **Path profiling**
  - **Reduce profiling overhead**

Summary

- **Moore's Law is still valid for the next 10+ years**
- **But significant challenges**
  - **Power**
  - **Increasing performance efficiently**
- **Opportunity: Increase cooperation between compiler and microarchitecture**
- **Efficient JIT and Runtime for intermediate languages**
- **Move from Instruction-level parallelism to thread-level parallelism**

Microprocessor 2010

- At Least 100x Performance of the Pentium® 4 Processor
- 20 Ghz
- Multiple Processors on a Die
- Multiple Threads per Processor
- Specialized Processors to Accelerate Human-Interface and Communications

Cooperation between Industry & Academia

- **Open source is a good way to coordinate research activities between industry and academia**
  - **Intel MRL open source Computing Vision Lib**
  - **Intel MRL open source Open Runtime Platform**
  - **Several other Intel open source utilities http://developer.intel.com/software/opensource**
- **Check out MRL web pages and Intel tools and compilers**

http://intel.com/research/mrl

http://developer.intel.com/vtune