

BOA: Targeting Multi-GHz with Binary Translation

**S. Sathaye, P. Ledak, J.
LeBlanc, S. Kosonocky, M.
Gschwind, J. Fritts, Z. Filan,
A. Bright, D. Appenzeller, E.
Altman, C. Agricola**

IBM

Presenter: **Erik Altman**

What is BOA?

- **BOA** =
 - ▶ **B**inary translation
 - ▶ **O**ptimized
 - ▶ **A**rchitecture

BOA Goals

- Execute existing *PowerPC* code 100% compatibly: ***User and supervisor state.***
- Execute at high frequency.
- => ***Use dynamic binary translation to map PowerPC code to code for high speed underlying machine.***

BOA Support for Binary Translation

- **64** regs vs **32** for *PowerPC*
- Extra bits with regs to help renaming, e.g. **CA** bit
- Ability to quash speculative I/O ops
- *Store Order Buffer* so can rollback to beginning of translation
- **LRA=Load Real Address** ins for crossing groups and pages

Outline of Talk

- Background
- BOA Software
- BOA Hardware
- Results
- Conclusions

Similarities of BOA and DAISY

- *PowerPC* target
- Use dynamic binary translation to achieve compatibility with *PowerPC*
- **VLIW / EPIC** style architecture

BOA and DAISY Differences (1)

BOA

- *PowerPC* ops from single path.
- **6 Issue**
- Ops assigned to FU's in pipeline
-
- **Stall-on-use**
- Memop sequence #'s, Address Comparators

DAISY

- *PowerPC* ops from multiple paths.
- **8-16 Issue**
- Mini-Icache maps fixed cache locations to FU's
- **Stall-on-miss**
- Load-Verify Instructions

BOA and DAISY Differences (2)

BOA

- Predicated bundles of 3 ops
- ***1 branch per cycle***
-
- Branch prediction

DAISY

- Tree instructions
-
- ***Up to 3 branches per cycle***
- Encode successor cache line in instruction => Fetch known ins each cycle

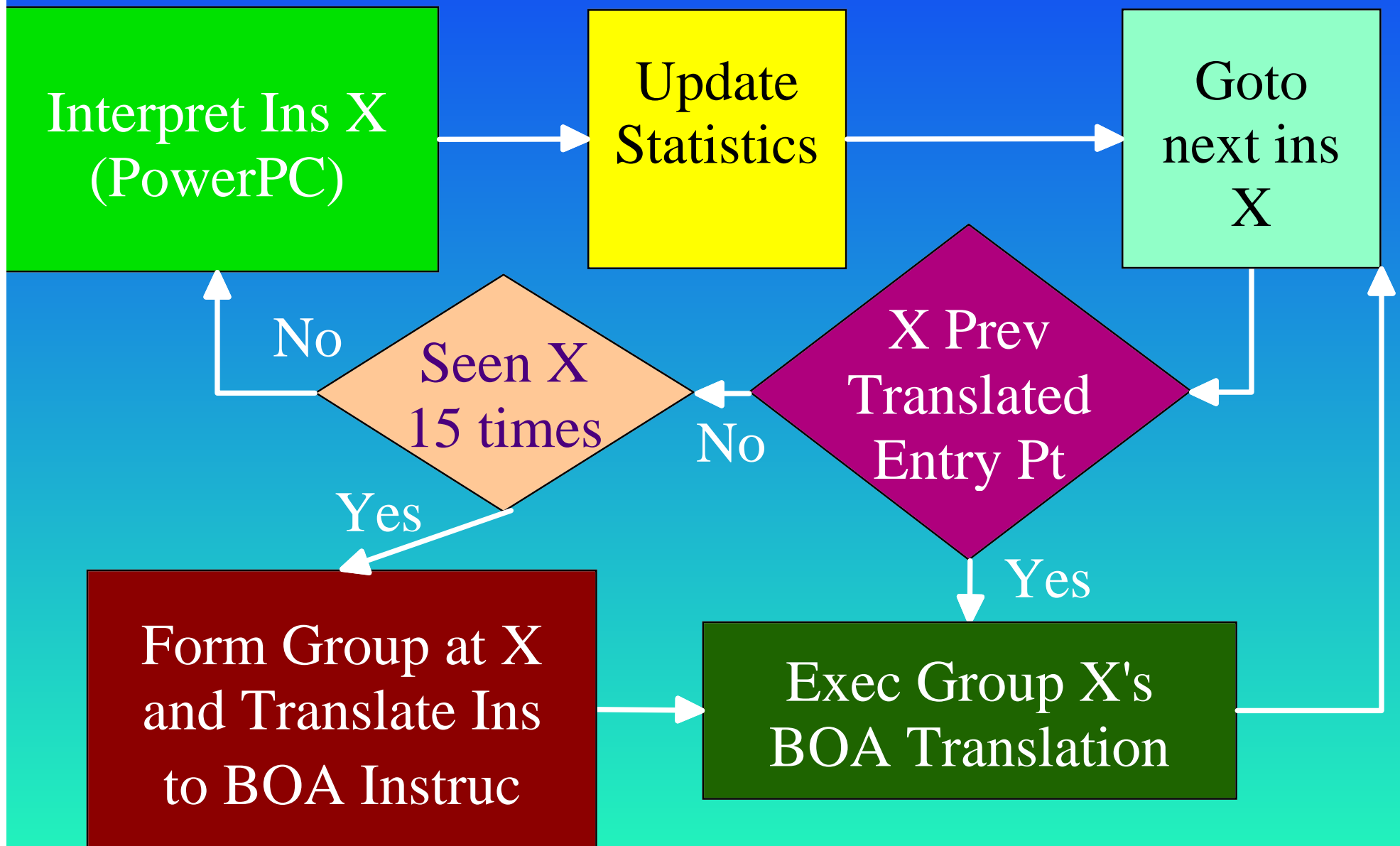
BOA and Other Binary Translation Approaches

- Runs whole architecture and not dependent on OS, unlike *FX!32*.
- Produces code for a different underlying architecture, unlike *Dynamo*.
- Focus on one architecture, unlike *Dixie* or *UQBT*.
- Unlike *Java JITS*:
 - ▶ Language Independent
 - ▶ But not portable across platforms

Why software dynamic translation?

- Hardware cracking consumes time and/or transistors
- Microcode emulation limits exploitation of ILP
- Software translation avoids these problems but requires high instruction reuse: ***Most apps have high reuse.***
- Dynamic translation allows 100% compatibility => ***Can handle arbitrary entry points, self-modifying code, etc.***

BOA System Architecture



Statistics

- For each *conditional branch*, keep *taken* and *fall-thru* counts
- For each *register branch*, keep list of register target values.
- Could keep load vals for *value prediction*

Group Formation

- Include ops from only a single path.
- Always follow most likely direction of conditional branch.
- If necessary, *invert branch conds* so all branches expected to fall-thru.
- Optionally go thru register branches:

```
cmpi cr15,PPC_LR,0x1234
```

```
bne EXIT_GROUP
```

```
# Translated Code from 0x1234
```

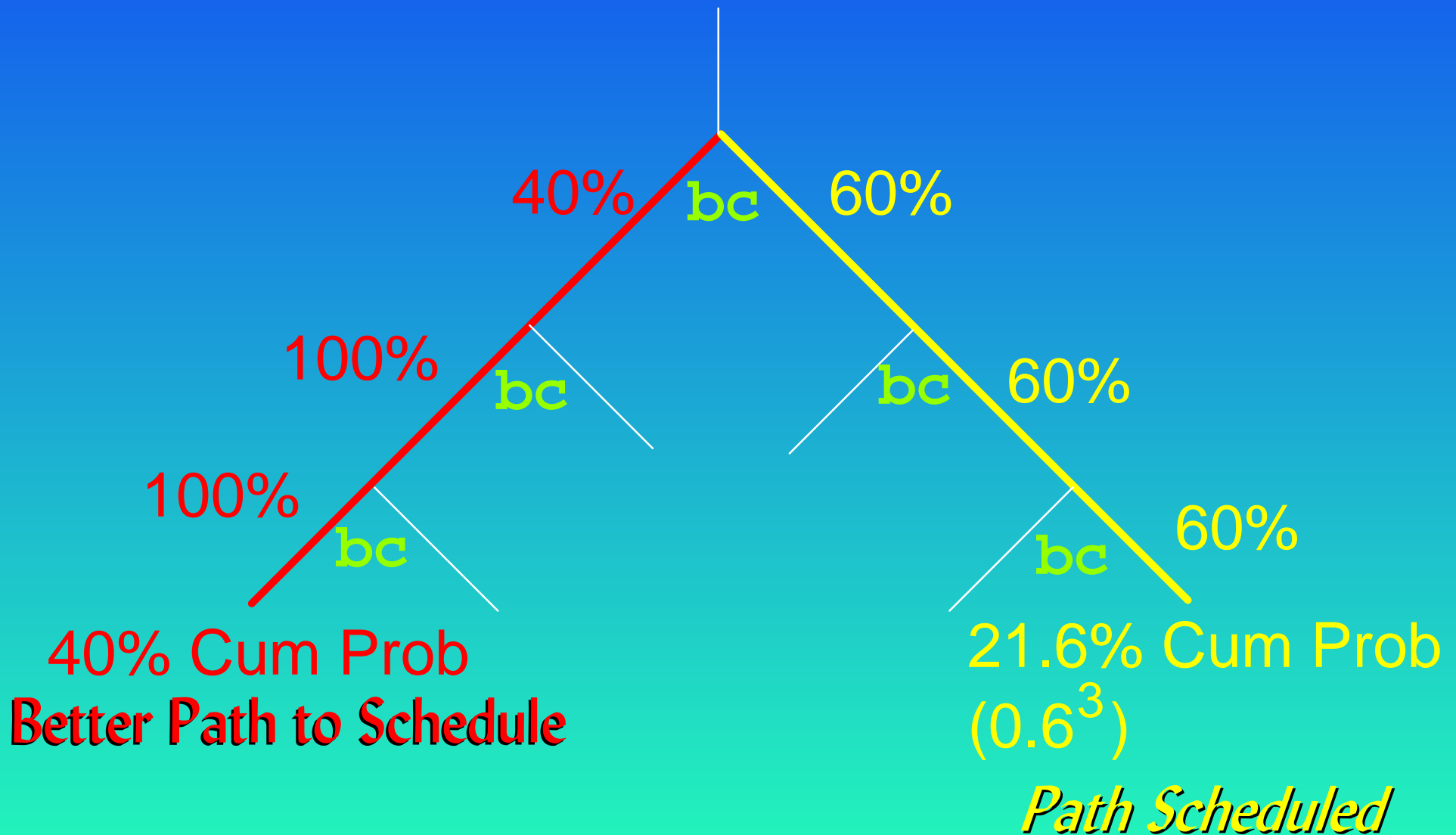
```
...
```

```
EXIT_GROUP: b BOA Syscode
```

Group Stopping Conditions

- Fickle branch:
 - ▶ E.g. go one direction less than **60%** of time
 - ▶ Termed **Bias-9** => One way **9/15** times
- # of ops in group > Threshold (60 ops)
- # of stores in group > Store Buffer Size
 - ▶ 32 entries in store buffer
- Register Branches (optionally)

Suboptimality



PowerPC State and Precise Exceptions

- At group entry, save PowerPC register state to shadow registers.
- Save done in one cycle by hardware when branching to new group.
- Copy values to PowerPC regs only at group exits.
- On exception:
 - ▶ Rollback to start of group
 - ▶ Restore PowerPC shadow registers
 - ▶ Interpret to find exception

PowerPC State and Precise Exceptions

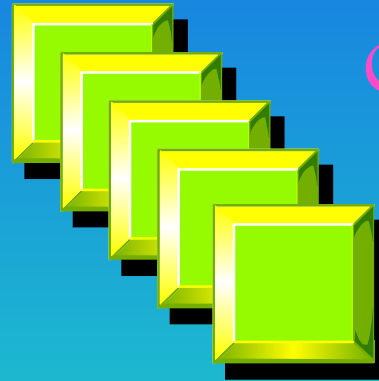
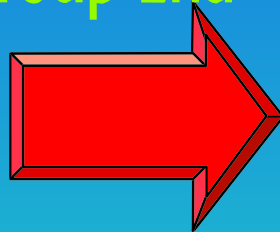
Scratch Regs

PowerPC Regs

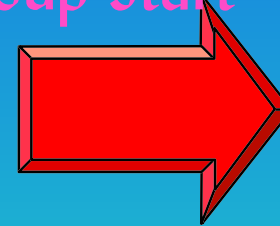
Shadow Regs



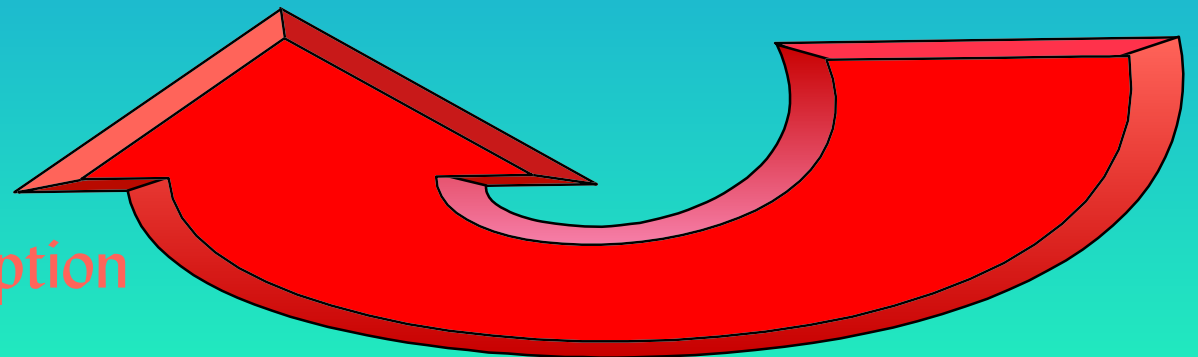
Group End



Group Start



Exception



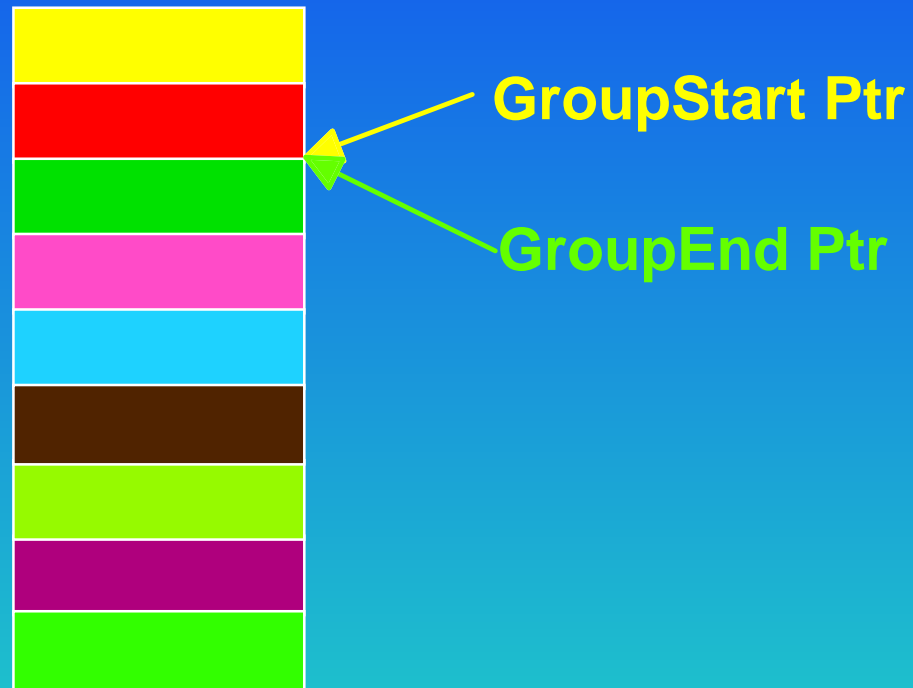
Support for STORES

- **Problem:** **STORE** executes but later instruction in group page faults.
- Want to rollback to group start, but must rescind all executed **STORES**.
- **Solution:** **Store Order Buffer (SOB)**
 - Stored values go to **SOB**, not memory
 - At group exit, all pending **STORES** in **SOB** are marked **eligible** for commit to memory.
 - **SOB** writes **eligible** values to memory in order.

Support for STORES

- **Problem:** **STORE** executes but later instruction in group page faults.
- Want to rollback to group start, but must rescind all executed **STORES**.
- **Solution:** **Store Order Buffer (SOB)**

Store Order Buffer (SOB)



SOB

Store Order Buffer (SOB)

BOA Group

STORE ○



GroupStart Ptr



GroupEnd Ptr



SOB

Store Order Buffer (SOB)

BOA Group

STORE ○

...

STORE ○



GroupStart Ptr

GroupEnd Ptr

SOB

Store Order Buffer (SOB)

BOA Group

STORE ○

...

STORE ○

...

Exception

=> Rollback



GroupStart Ptr

GroupEnd Ptr

SOB

Store Order Buffer (SOB)

BOA Group

STORE ○

...

STORE ○

...

STORE ○



GroupStart Ptr

GroupEnd Ptr

SOB

Store Order Buffer (SOB)

BOA Group

STORE

...

STORE

...

STORE

...

BSHAD



SOB

Memory

Speculative Load Support

- Use ctr to assign *sequence number* to each **LOAD** and **STORE** in a group.
- *Sequence number* part of opcode
- On **STORE**, hardware checks
 - **STORE** addr overlaps a prev **LOAD** addr
 - Prev **LOAD** addr has higher *sequence number* than **STORE**
- If aliasing:
 - Rollback group to start and re-execute
 - Possibly retranslate to unspeculate **LOAD**

Speculative Load Support (1)

- Use ctr to assign *sequence number* to each **LOAD** and **STORE** in a group.
- *Sequence number* part of opcode:

PowerPC Code

LOAD X

...

STORE Y

...

LOAD Z

...

BOA Group

1 LOAD X

3 LOAD Z

2 STORE Y

...

Speculative Load Support (2)

- **STORE** addr overlaps a prev **LOAD** addr
- Prev **LOAD** addr has higher *sequence number* than **STORE**

BOA Group

1 LOAD X

3 LOAD Z

2 STORE Y

...

Z aliases
with Y

Seq #3 >
Seq #2

Speculative Load Support (3)

- If aliasing:
 - ▶ Rollback group to start and re-execute
 - ▶ Possibly retranslate to unspeculate **LOAD**

Branching Between Groups and Across Pages

- Branch directly from group to group.
- Put **LRA** ins at start of each group and every page crossing.
- **LRA = Load Real Address:**
 - ▶ Get **real PowerPC address** from Virt Addr
 - ▶ Compare **real addr** to *real addr* when translation was done
 - ▶ Trap if mismatch or fault

BOA ISA (1)

- BOA is variable length VLIW machine.
- BOA instructions (bundles) are 128 bits.
- Bundles have 3 primitive ops.
- Primitive ops have 39 bits plus stop bit.
- 8 bits of bundle reserved for future uses such as predication.
- Instruction Issue:
 - ▶ Up to 6 primitive ops are issued together.
 - ▶ Only last op issued may have stop bit set.

BOA ISA (2)

- **64** Integer Registers
- **64** Float Registers
- **16** *4-bit* Condition Registers
- Branches take **1** cycle:
 - ▶ Branch mispredicts cost **7** cycles
 - ▶ Static branch pred (*using interpreter stats*)
 - ▶ At most one branch per cycle

BOA Latencies

- Integer ops take **1** cycle
 - ▶ **No bypass** => *Dependent ops must be 2 cycles apart*
- LOADs take **3** cycles
 - ▶ **No bypass** => *Dependent ops must be 4 cycles later*

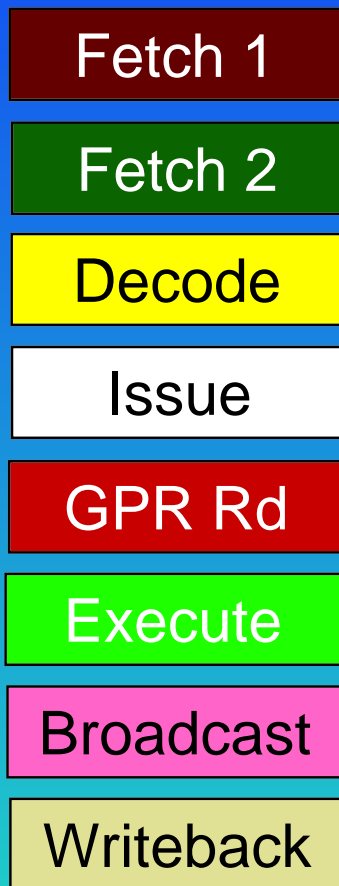
BOA Resources

- **6 *Issue* Slots**
- **2 *LOAD / STORE* units**
 - ▶ Each with own copy of register file
- **4 *Integer* units**
 - ▶ Each with own copy of register file
- **2 *Float* units**
- **1 *Branch* unit**
- **32-entry *Load* and *Store Buffers***
- **Register scoreboarding of LOAD values**
 - ▶ Stall when try to use loaded value

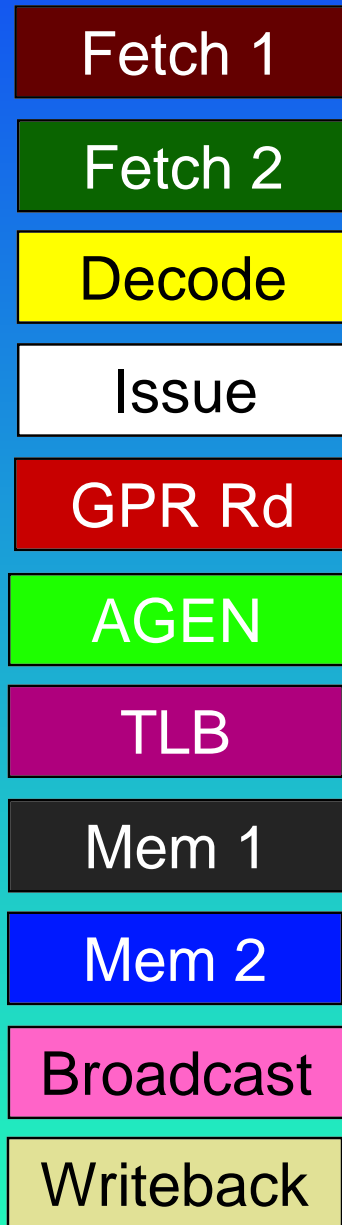
BOA Frequency

- BOA has **700 ps** worst case cycle time in current 0.18 um CMOS bulk process
- **700 ps** is more than 50% better than other 0.18 um designs scaled from 0.25 um
- SIA Roadmap: FET's 1.8x faster by 2005
 - ▶ BOA should be **400 ps (2.5 GHz)** by 2005
 - ▶ BOA has extra pipe stages to account for wire delay

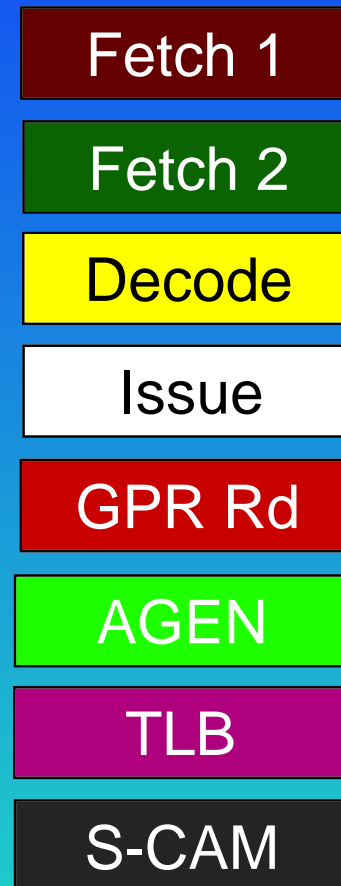
BOA Pipelines



Integer

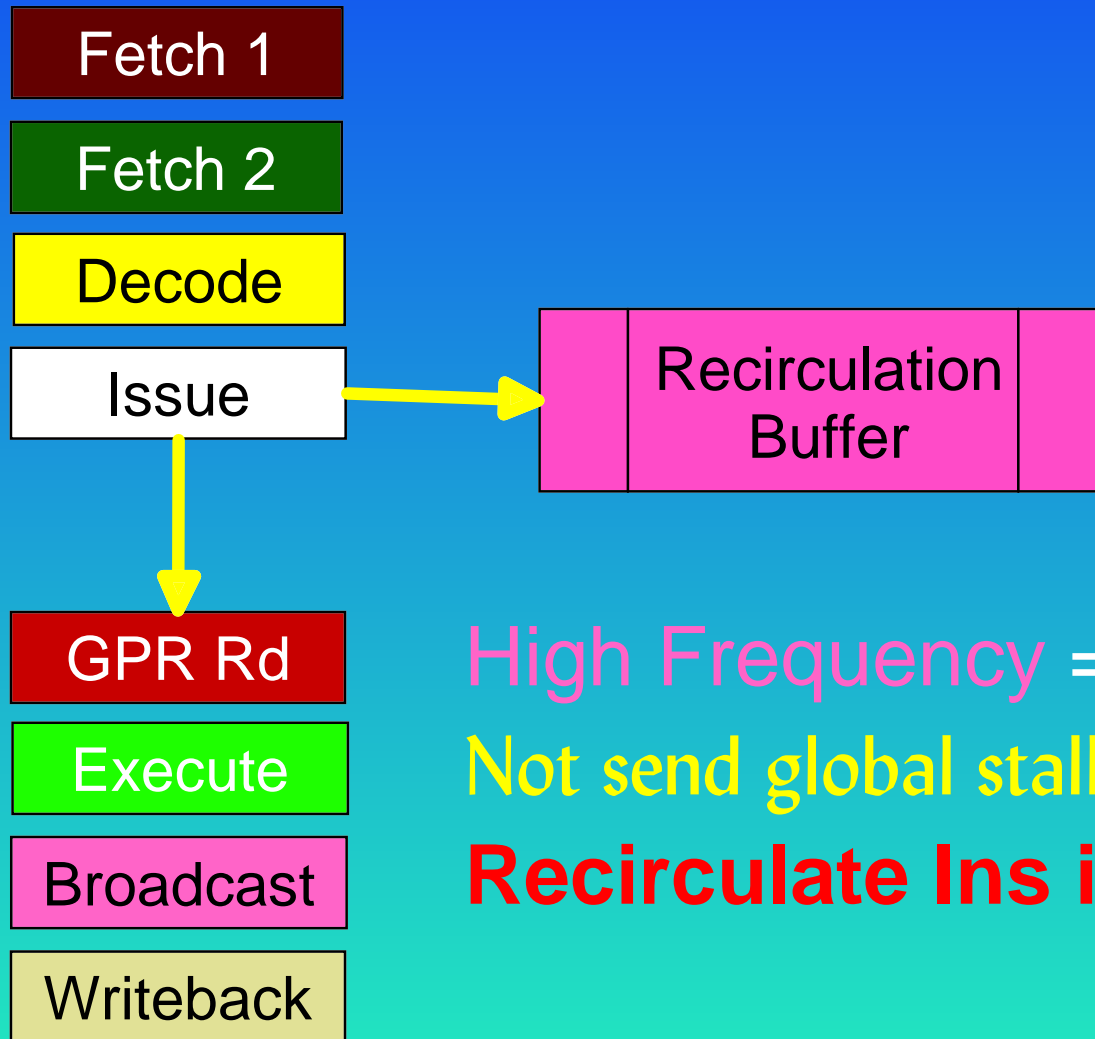


LOAD



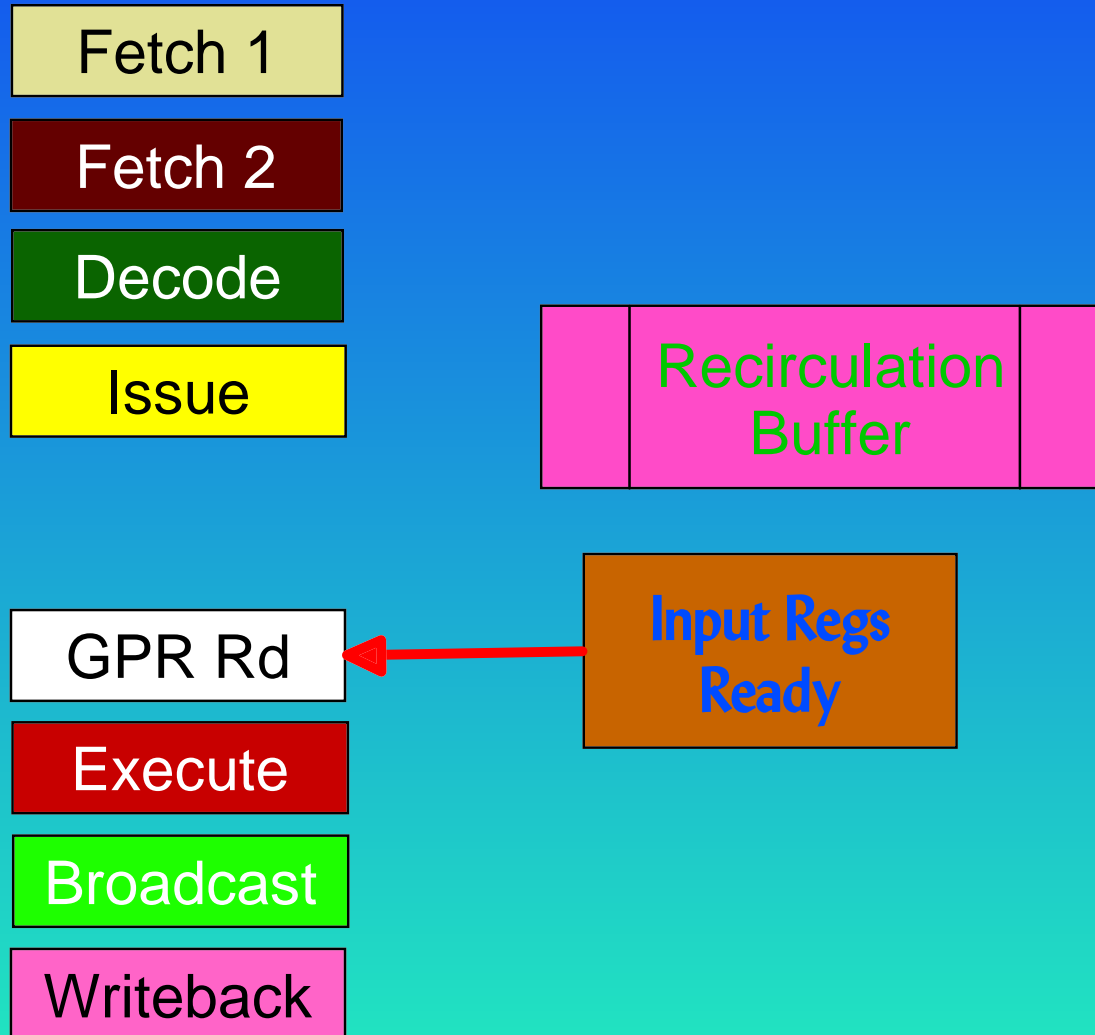
STORE

Recirculation

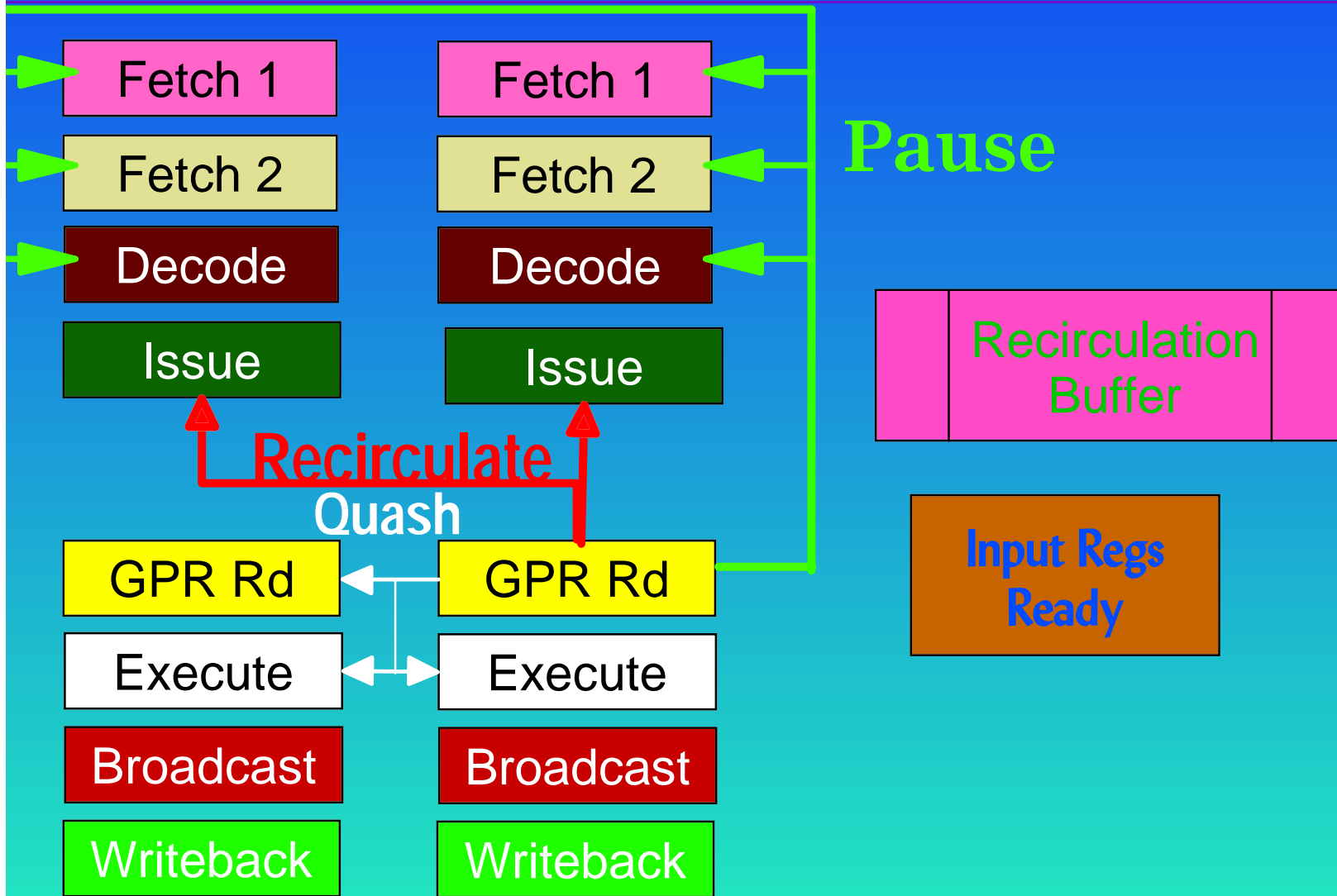


High Frequency =>
Not send global stall signals.
Recirculate Ins instead.

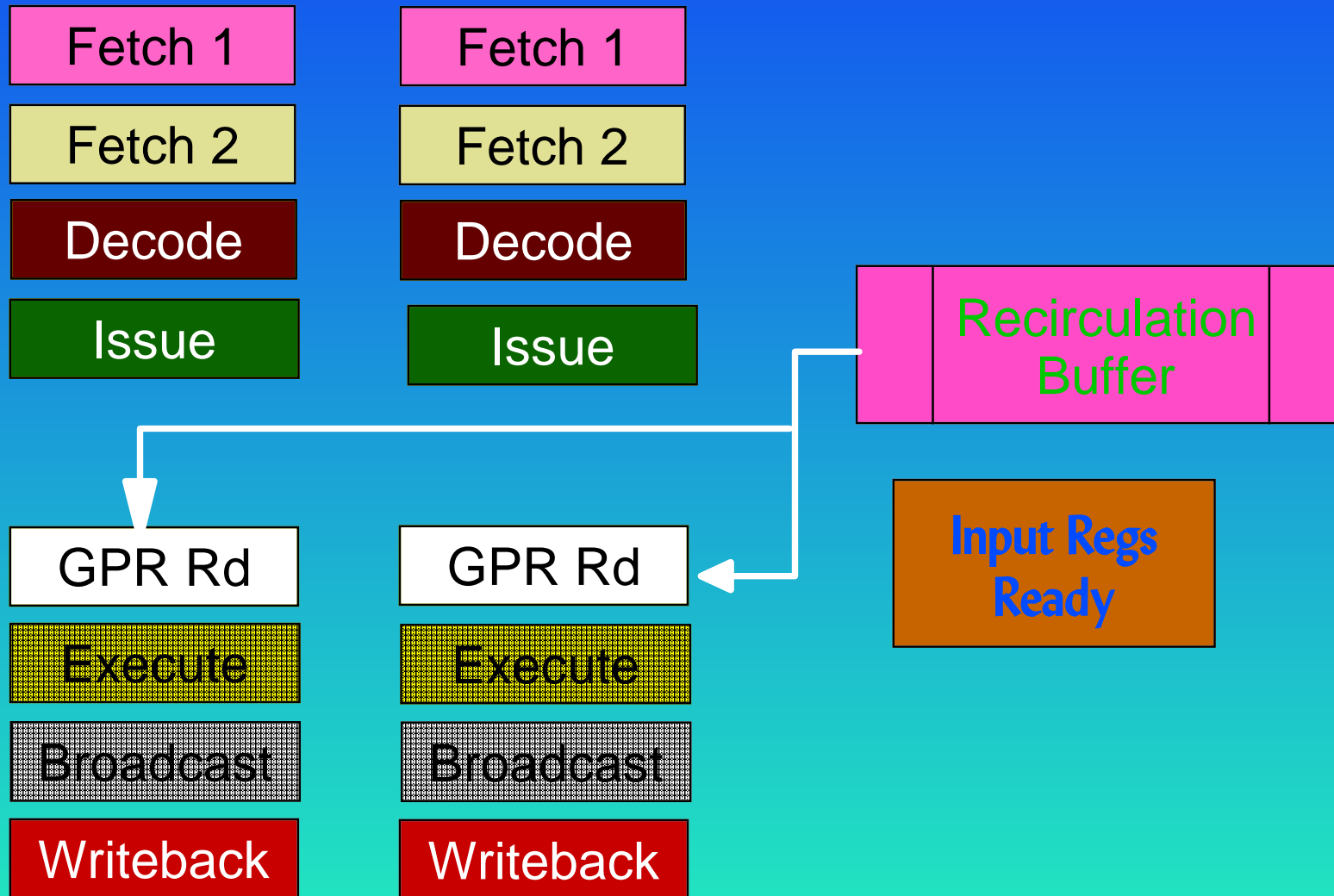
Recirculation



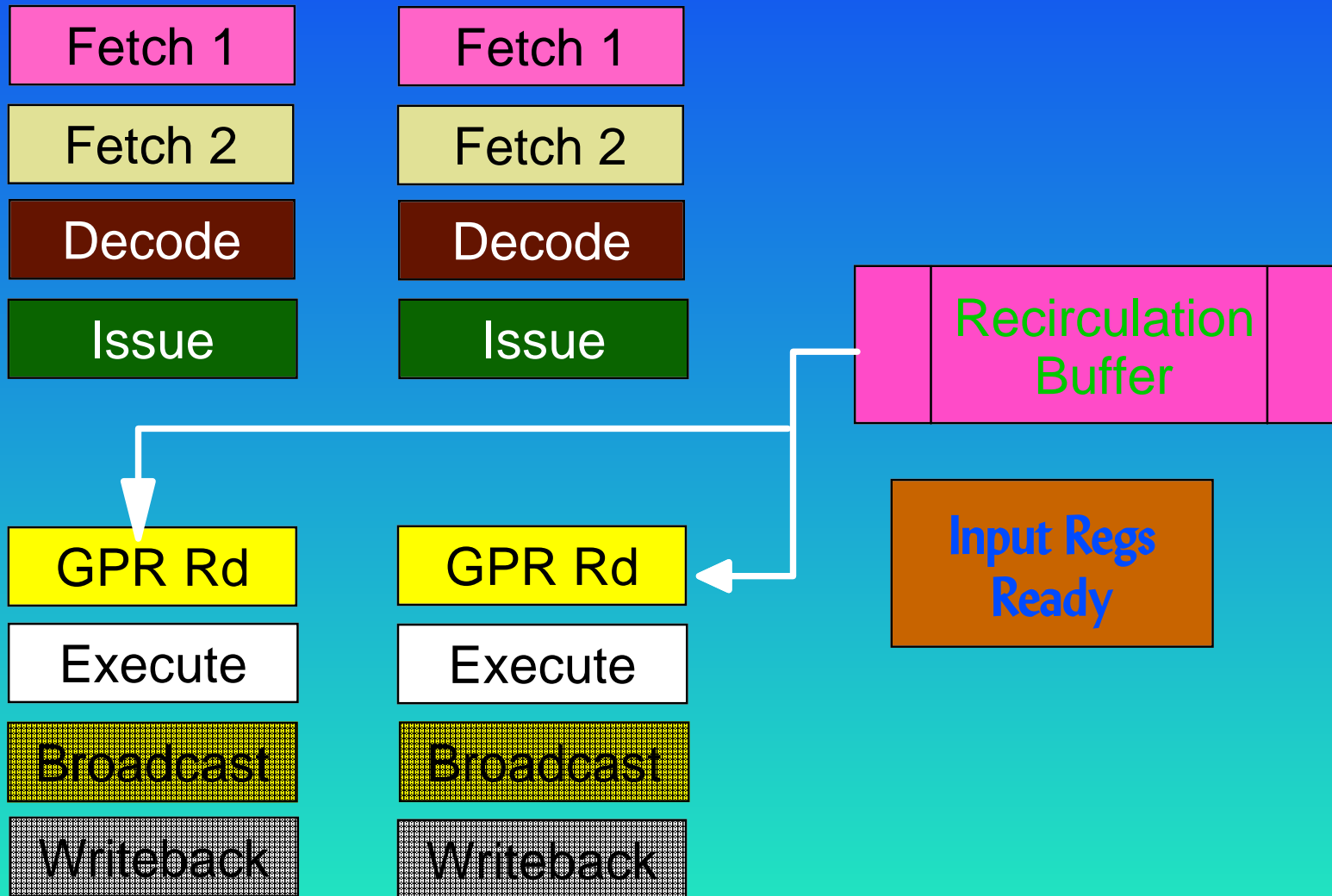
Recirculation



Recirculation



Recirculation



Recirculation

Fetch 1

Fetch 2

Decode

Issue

GPR Rd

Execute

Broadcast

Writeback

Fetch 1

Fetch 2

Decode

Issue

GPR Rd

Execute

Broadcast

Writeback

Recirculation Buffer

Input Regs Ready

Recirculation

Fetch 1

Fetch 2

Decode

Issue

Fetch 1

Fetch 2

Decode

Issue

Recirculation Buffer

GPR Rd

Execute

Broadcast

Writeback

GPR Rd

Execute

Broadcast

Writeback

Input Regs Ready

Recirculation

Fetch 1

Fetch 2

Decode

Issue

GPR Rd

Execute

Broadcast

Writeback

Fetch 1

Fetch 2

Decode

Issue

GPR Rd

Execute

Broadcast

Writeback

Recirculation Buffer

Input Regs Ready

BOA Caches

	<i>Size</i>	<i>Line Size</i>	<i>Assoc</i>	<i>Hit Latency</i>
L1 - Ins	256K	256	4	1
L1 - Data	64K	128	2	4
L2 - Joint	4M	128	8	14
Memory				90

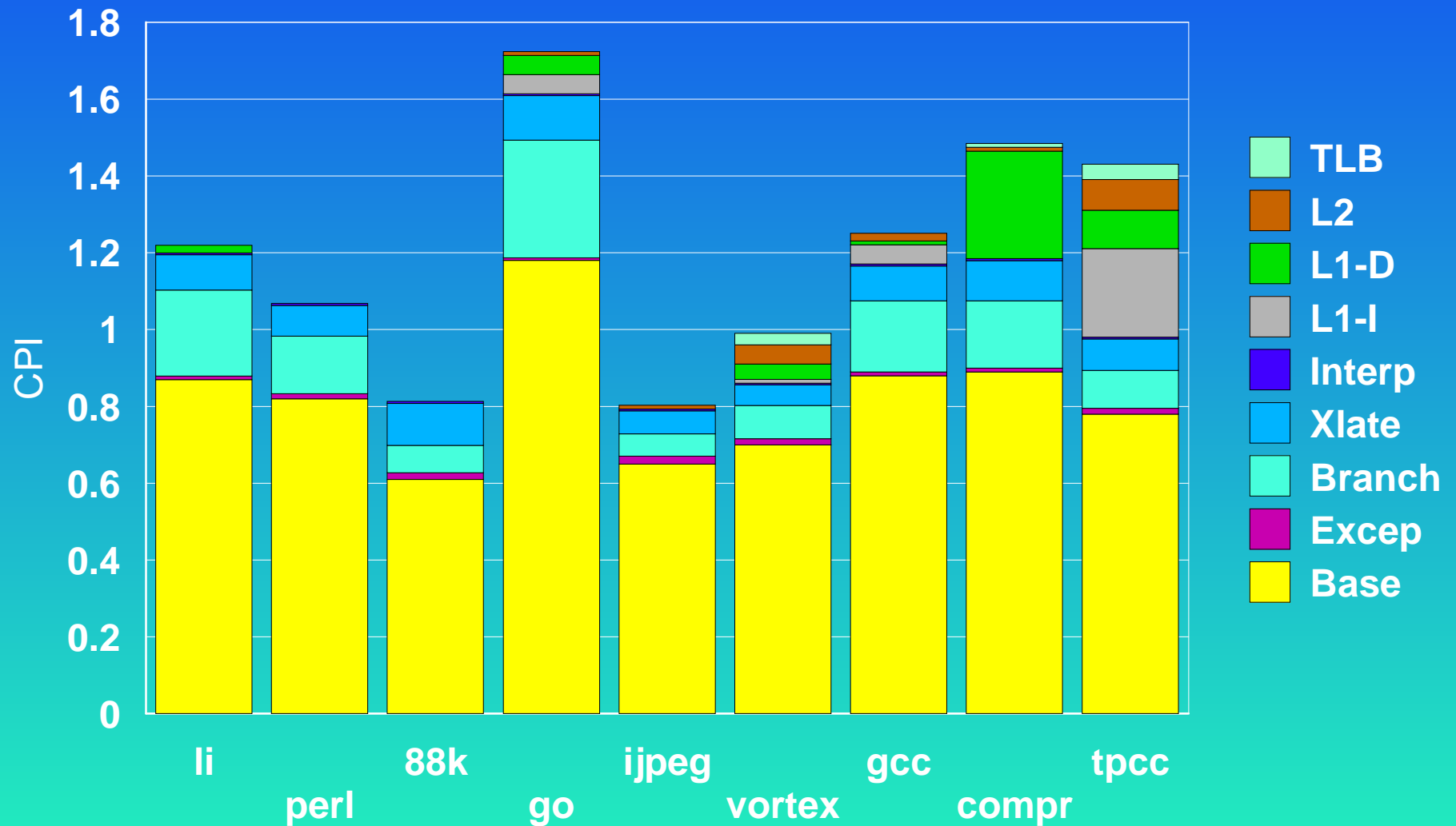
Benchmarks

- **Benchmarks**
 - *SPECint95*
 - *TPC-C*
- **SPECint95 Sampling Method**
 - Uniformly Sampled PowerPC Traces
 - **2 million** instructions per sample
 - **50** samples per benchmark
- **TPC-C Sampling Method**
 - Special-purpose hardware
 - **170 million** instruction trace

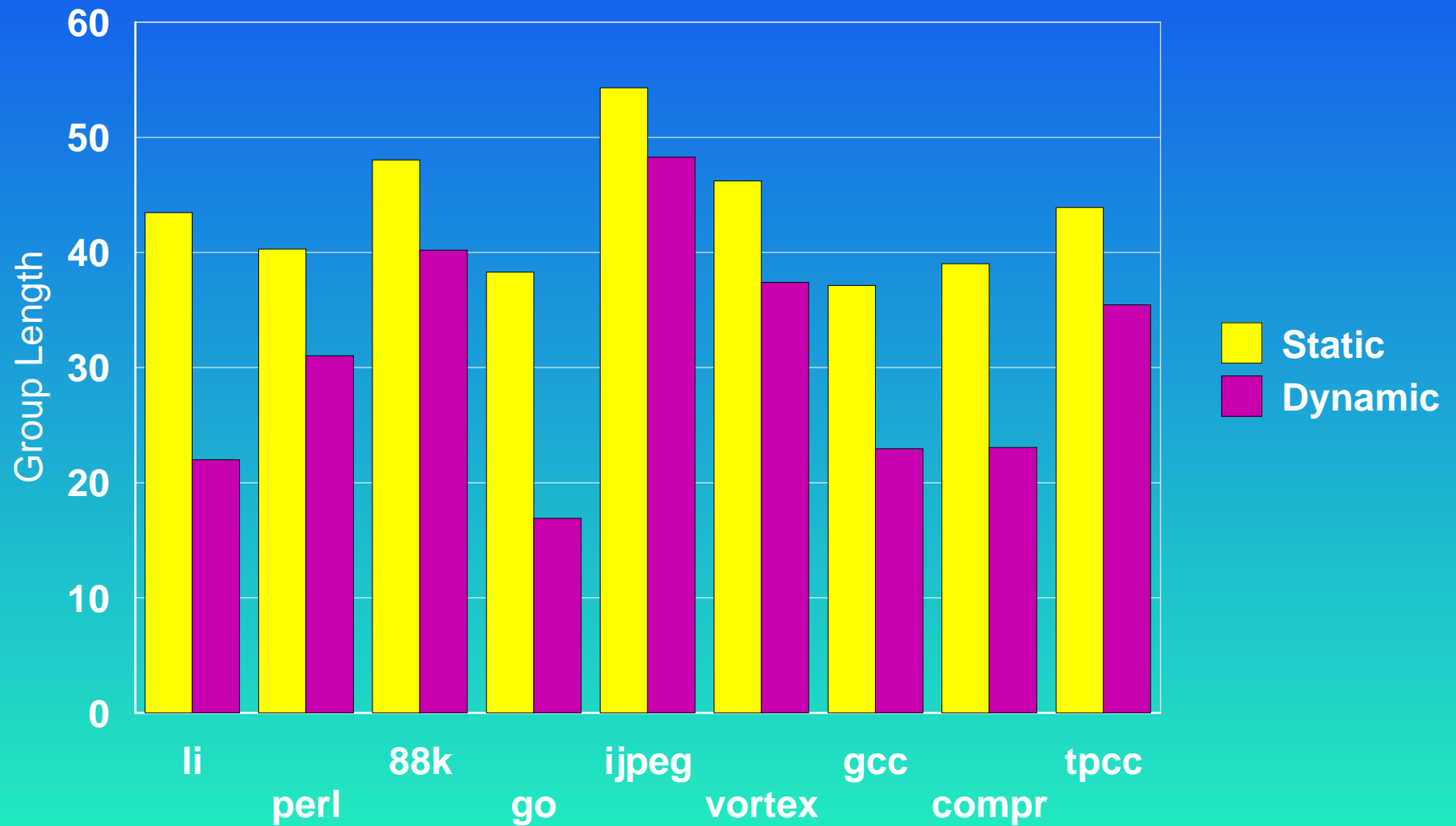
Factors in BOA Performance

- **Instruction Reuse Rate**
- **# of times each ins is translated**
- **Translator CPI**
- **Interpreter CPI**
- **Statistics CPI**
- **Synchronous Exception Rate**
- **ICache flushing from translator**
- **Average Group Length**
- **# of times interpret before translating**

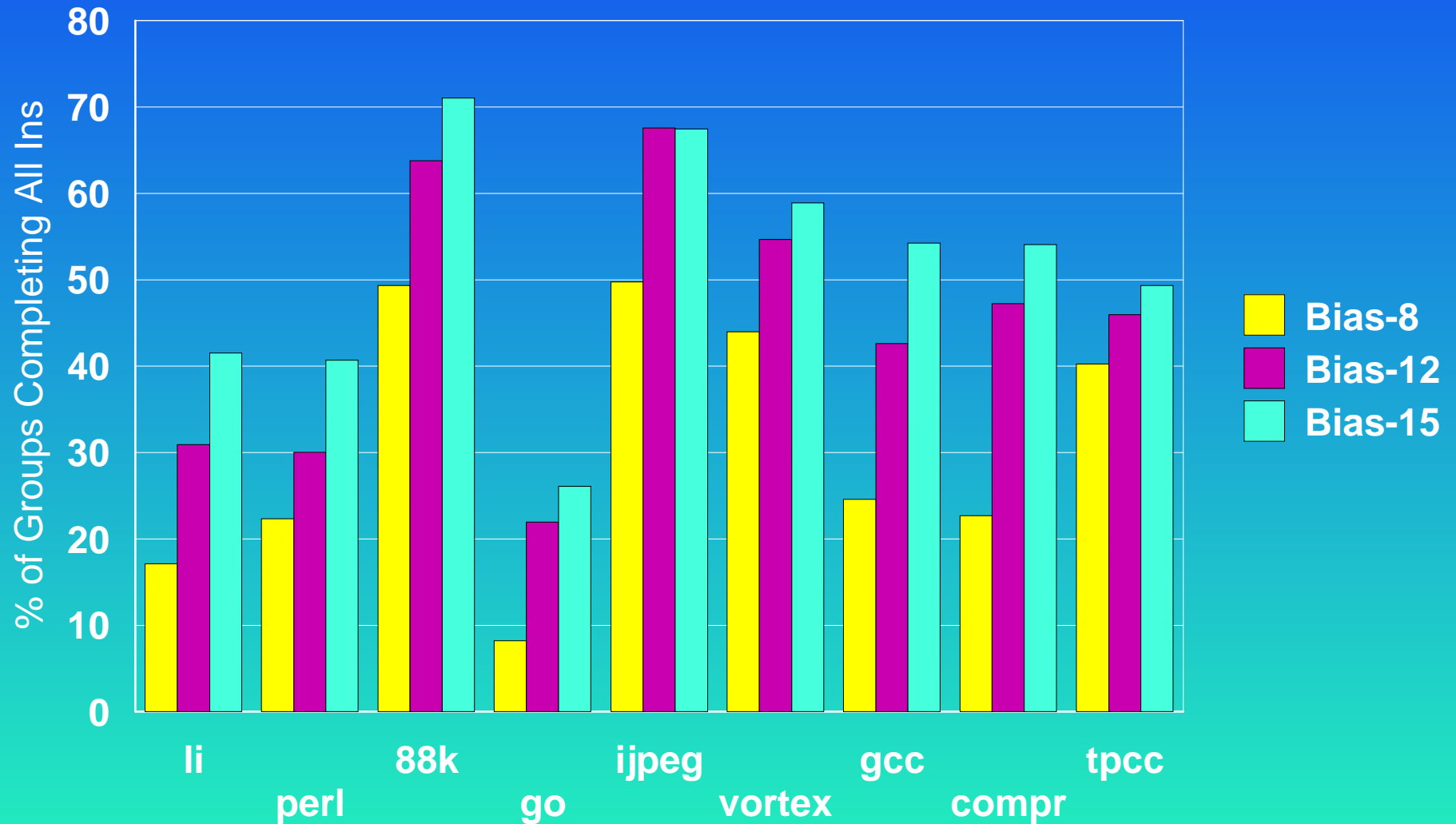
BOA Baseline CPI



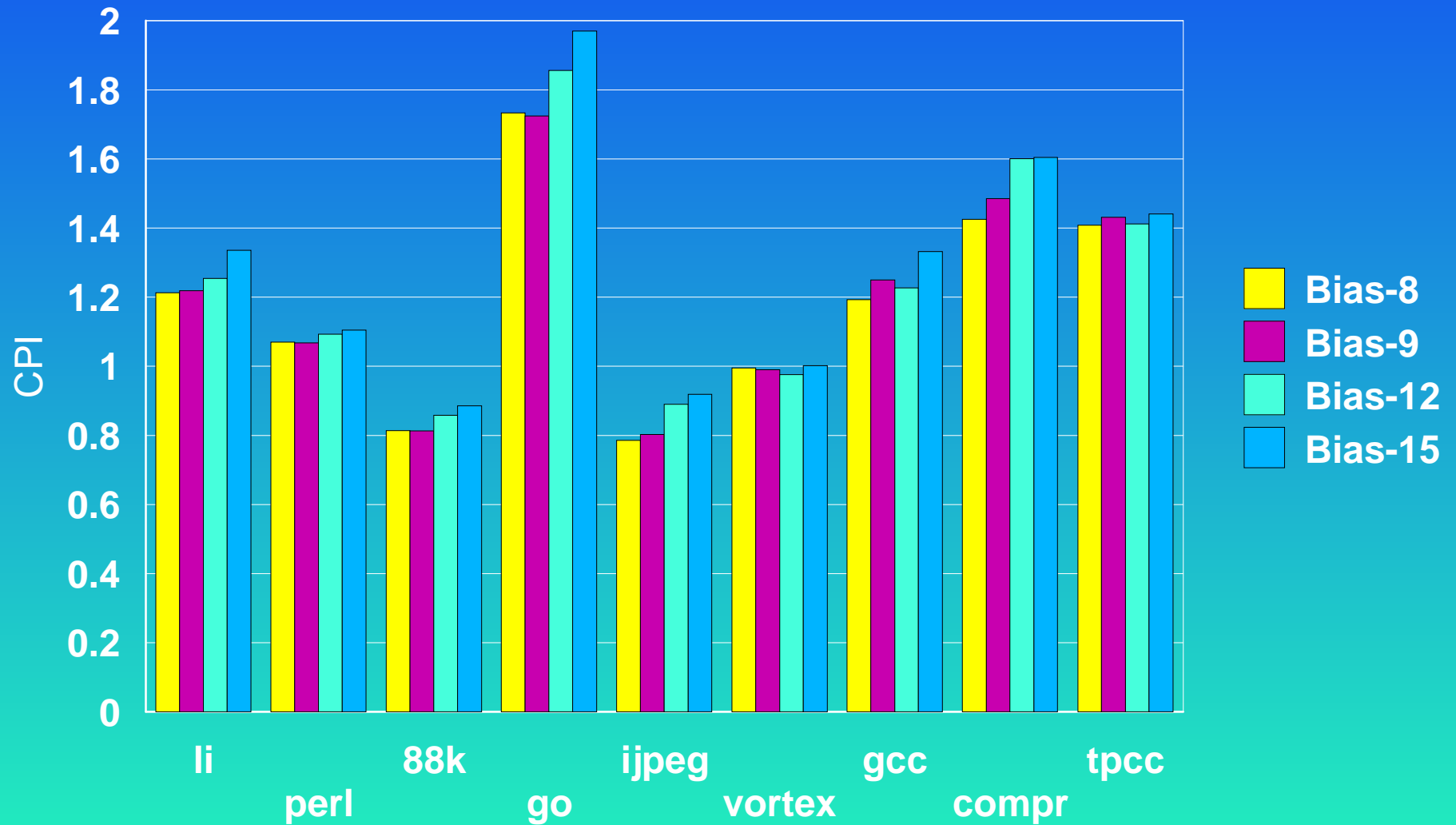
BOA Group Lengths



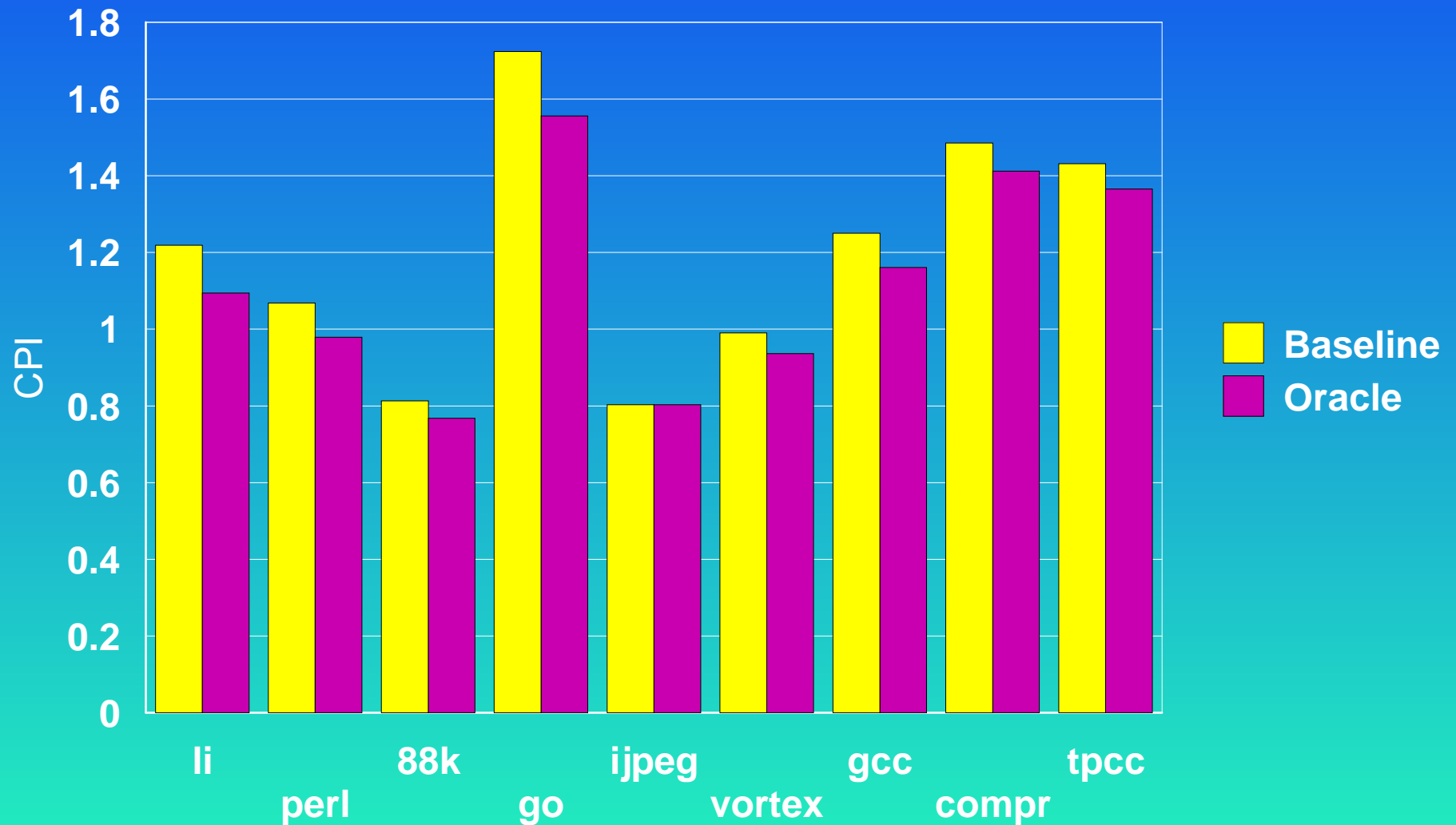
% of Groups Completing All Ins



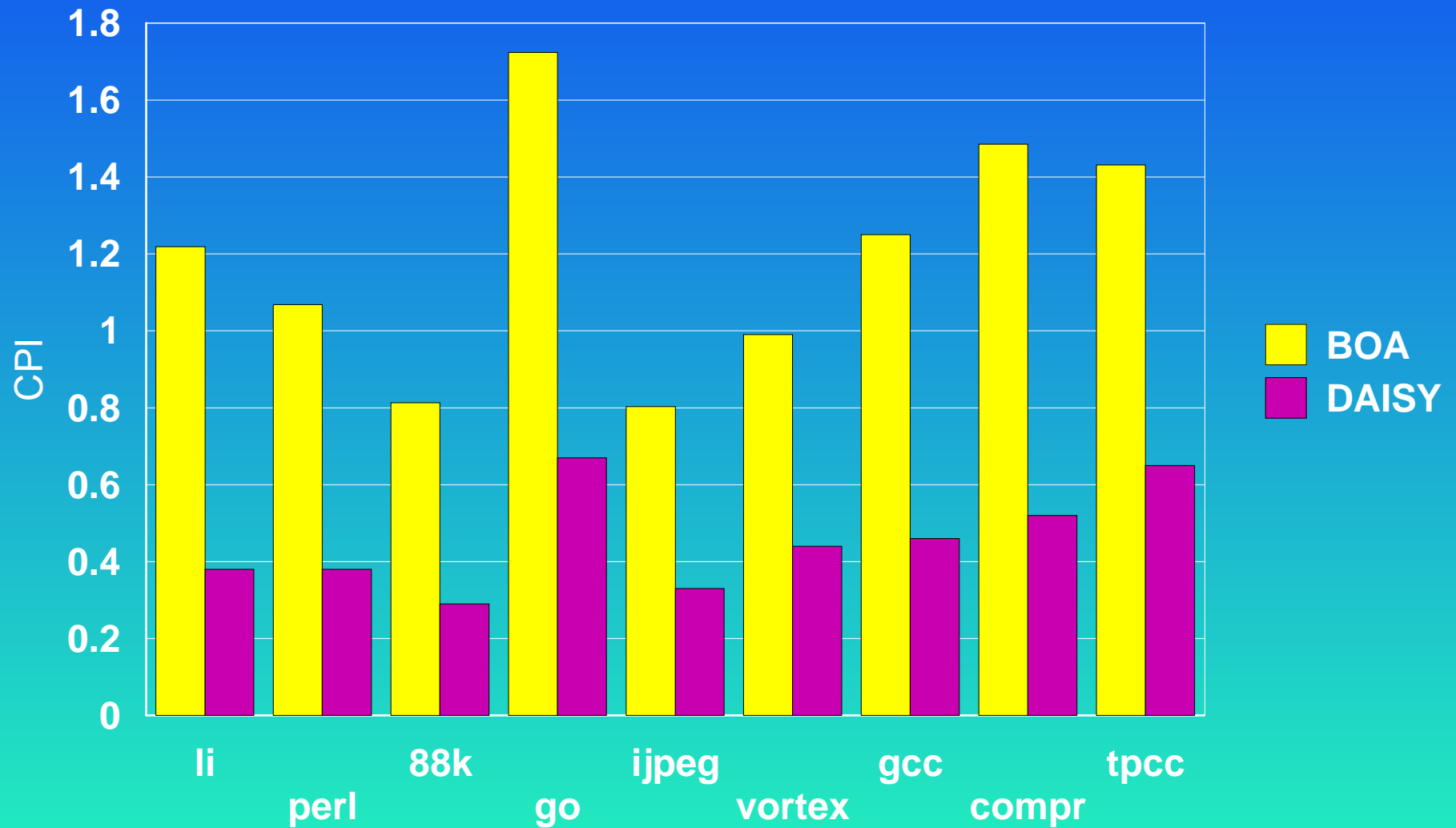
Effect of Bias on CPI



Oracle Static Branch Prediction



BOA and DAISY CPI



Conclusions

- **BOA** uses dynamic binary translation to fully emulate the *PowerPC* architecture.
- **BOA** is a VLIW architecture designed for high frequency operation.
- **BOA's** CPI of around 1 is comparable to most existing superscalars.

Future Work

- Multipath Scheduling:
 - ▶ Without *frequency loss*
 - ▶ Without *code explosion*
- More use of *profiling* and *value prediction*
- Reductions in *Translation Cost*