

# **Inherently Lower Complexity Architectures using Dynamic Optimization**

**Michael Gschwind  
Erik Altman**

**IBM T.J. Watson Research Center**

# What is the Problem?

- Out of order superscalars achieve high performance.
- ... **But at the cost of** *high hardware complexity*
  - ▶ Predictors
  - ▶ Complex decode
  - ▶ Complex issue queues with wakeup and issue logic
  - ▶ Register mapping tables
  - ▶ ...

# What is the Problem?

- Out of order superscalars achieve high performance.
- ... **But at the cost of *high power*.**
  - ▶ Many out of order components operate every cycle.
  - ▶ Many components query a large set of data to operate on a single element.

# What is the Problem?

- **Out of order superscalars achieve high performance.**
- **... But at the cost of *deep pipelines*.**
  - ▶ Complex logic has long latency.
  - ▶ To achieve high frequency with long latency, super pipelining is required.
  - ▶ Deep pipelines require excellent branch predictors.
  - ▶ Excellent branch predictors are complex.
  - ▶ Complex logic has long latency ...

# What is the Problem?

- Out of order superscalars achieve high performance.
- ... **But at the cost of** *high verification and debug complexity.*
  - ▶ With Moore's Law, schedule slips = performance slips

Schedule Slip	Relative Performance
1 month	4%
3 month	12%
6 month	26%
9 month	41%
12 month	59%
18 month	100%

# What is the Solution?

- **Software Dynamic Optimization**
- **Allows reduced hardware complexity:**
  - ▶ Shorter pipelines for same frequency.
  - ▶ Fewer hardware predictors.
  - ▶ Simpler issue logic.
  - ▶ Less power, a la Transmeta.
  - ▶ Less debug and verification.
  - ▶ Smaller chips and higher yield.

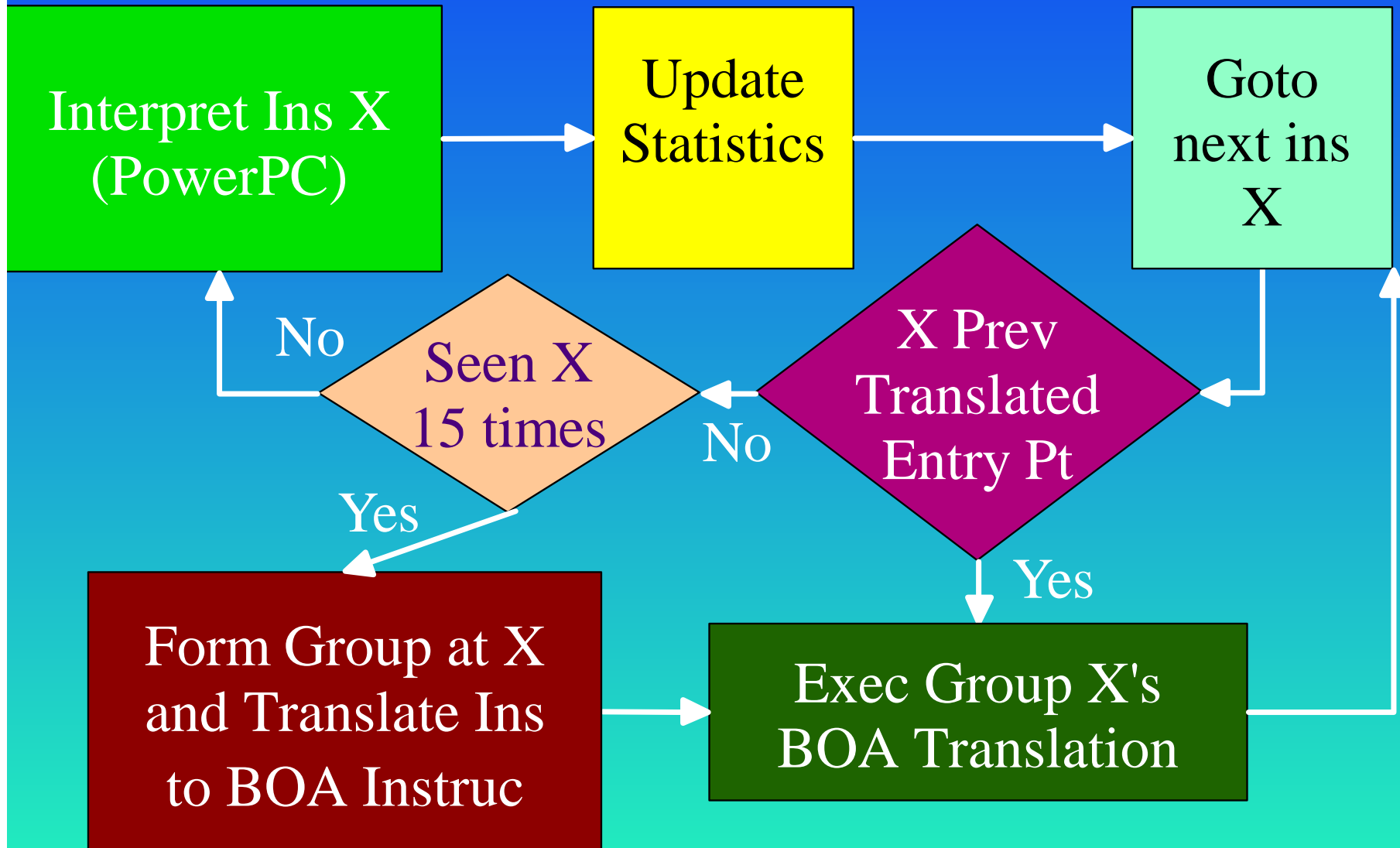
# How to Implement the Solution

- BOA Architecture for Complexity Effective Design
- **BOA** = **B**inary Translation **O**ptimized **A**rchitecture
- BOA in combination with its dynamic optimization software is architecturally compatible with PowerPC.

# What is interesting about BOA?

- Software dynamic optimization.
- Precise behavior on most memory faults.
- Load/Store order tables ensure memory semantics and allow aggressive dynamic software reordering.
- Instruction recirculation mechanism to simplify issue and exception handling.
- Predictable latencies handled by software, unpredictable by hardware.

# BOA System Architecture



# 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.
- Complex PowerPC ops cracked.
- 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**

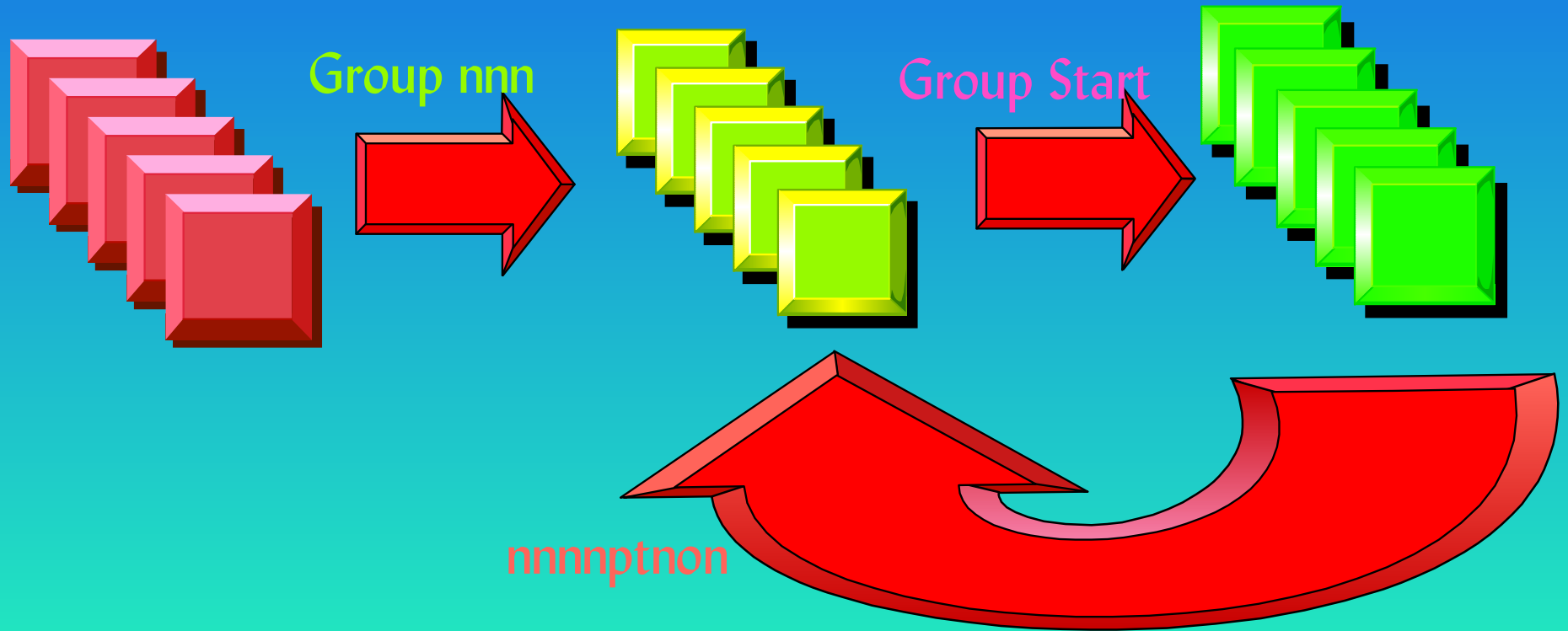
**Instructions**

## 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

# PowerPC State and Precise Exceptions

Scratch Regs    PowerPC Regs    Shadow Regs



# 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

# Dynamic Optimization

# BOA Dynamic Optimization

- BOA's software optimizer originates with IBM's earlier DAISY project.
- BOA adjusted and tuned optimizer:
  - ▶ To support a narrower, higher frequency target machine.
  - ▶ To optimize along single hyperblock paths, instead of tree region with multiple paths.
    - Improves code packing, reduces TLB misses
    - Improves code layout and helps IFetch, a la trace caches.

# Dynamic Optimization Environments

- **Dynamic Optimization can be used in a variety of environments:**
  - ▶ **Process level**
    - Idealized virtual memory
    - Fewer difficult system/kernel code issues
  - ▶ **Operating system level**
    - No modifications to operating system
    - More transparent
      - Less danger of compatibility issues

# Dynamic Optimization Targets (1)

- **Simpler implementation of the same architecture**
  - ▶ **Ability to bail out and revert to native execution:**
    - **If overhead too high**
    - **For hard to emulate sequences**
    - **When no benefit of DO can be measured**
      - **Or actually degrades**

# Dynamic Optimization Targets (2)

- Different architecture, e.g., RISC => VLIW
  - ▶ Drastically simplify architecture
  - ▶ Reduce decoding overhead even further
  - ▶ Add more registers, add new concepts
    - All code must be emulated. Can cause severe degradation if low reuse, e.g. WinStone.
  - ▶ Get benefits of code packing

# Some Optimizations

- Code packing
- Register Port arbitration
- Exploit novel architecture concepts
- Improve predictability of execution path by code layout
- Eliminate performance-degrading ops
- Avoid use of complex idioms, e.g. condition register broad side read/write
- Replace with easier to schedule/execute ops

# **Code Packing - Software Based Trace Caching**

- **Code packing:**
  - ▶ **Application-directed code compaction**
  - ▶ **Similar concept to hardware trace cache**
  - ▶ **Much simpler to implement**
  - ▶ **Increase effectiveness of ICache and ITLB**
  - ▶ **Very helpful in HP Dynamo performance**

# Dynamic optimization and architecture styles

Technique	OOO	DO+OOO	DO+IO	DO+VLIW
ISA	base ISA	base ISA	base ISA	new ISA
general optimizations	too complex	DO optimizes	DO optimizes	DO optimizes
path-predictive fetching	1 fetch prediction	DO improves prediction	DO improves prediction	DO improves prediction
code compaction	trace cache	DO performs layout	DO performs layout	DO performs layout
select insns to issue	wakeup/select logic	wakeup/select logic	DO adapts at exec. time	DO adapts at exec. time
precise exceptions	register renaming	register renaming	SW recovery code	SW recovery + HW support
complex insns	decoder cracks	decoder cracks	DO or HW	DO cracks and layers
form issue groups	select logic	select logic	issue logic	DO groups packets

**BOA**

**Processor**

# 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

# Additional Hardware Simplification from Dynamic Optimization

- **Dynamic optimizer:**
  - ▶ Limits number of register read/write ports, with little performance effect.
  - ▶ Handles PowerPC quirks, e.g.
    - Condition register used as 32-bit value, 8x4-bit values, and as 32x1-bit values. Renaming must account for all cases.
    - PowerPC addressing treats register R0 as literal 0. BOA addressing treats all registers uniformly.

# 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
...		

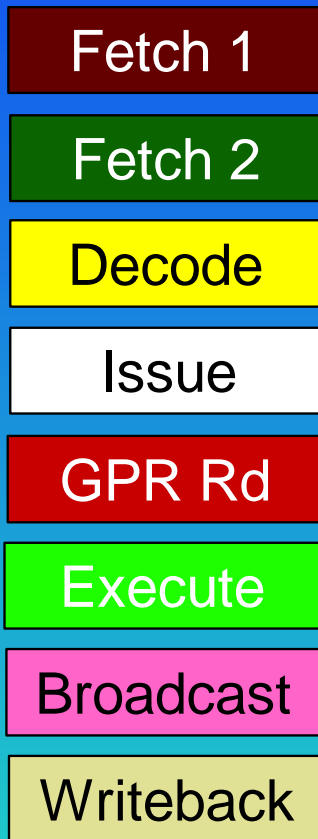
Z aliases  
with Y

Seq #3 >  
Seq #2

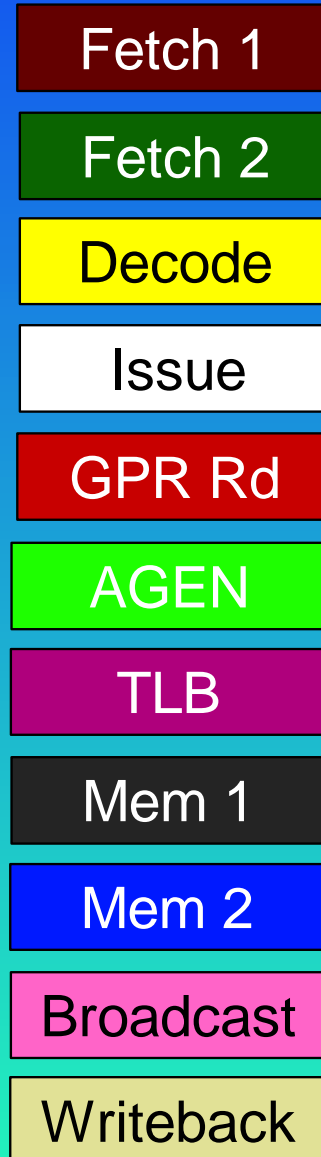
# Speculative Load Support (3)

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

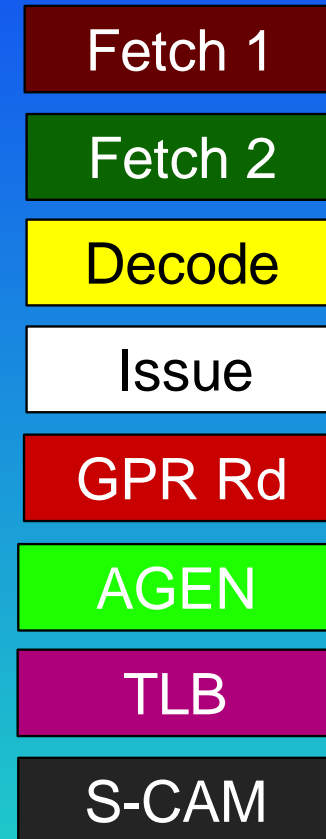
# BOA Pipelines



**Integer**

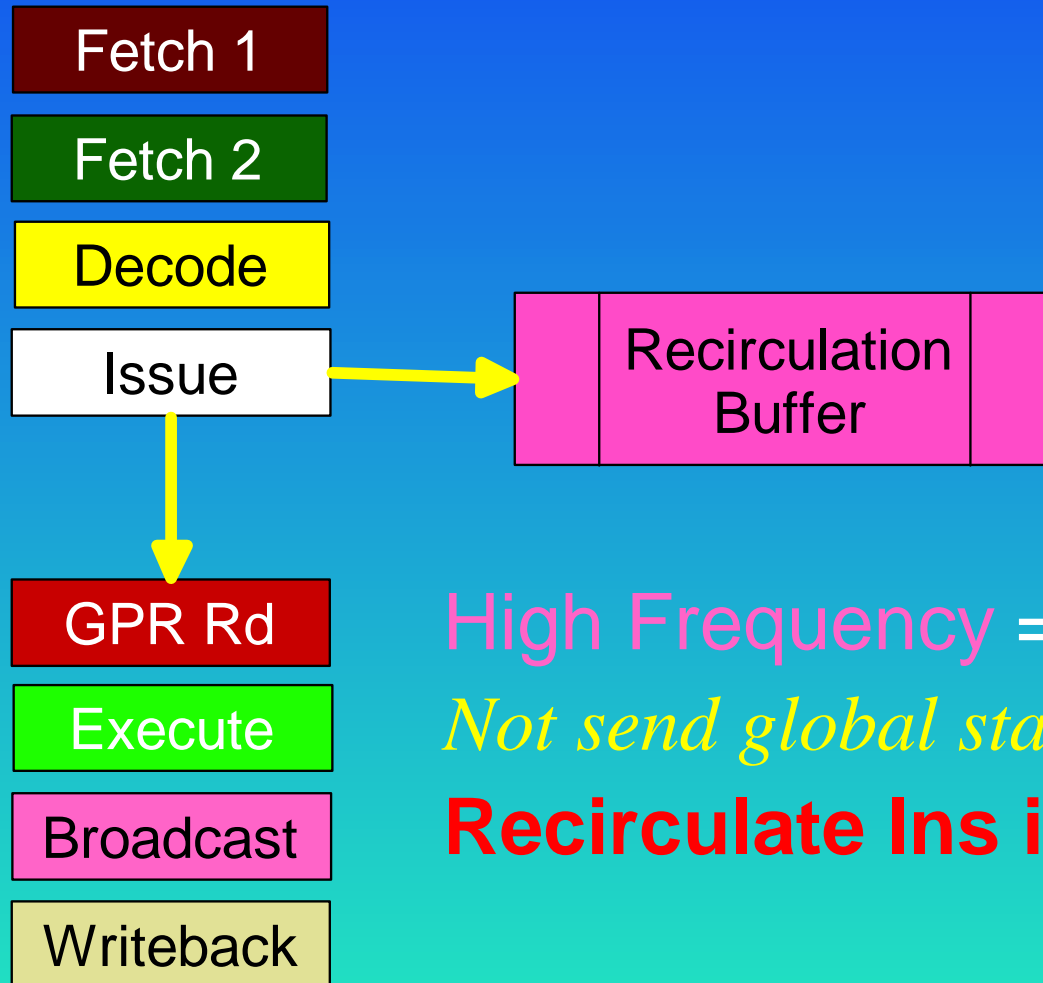


**LOAD**



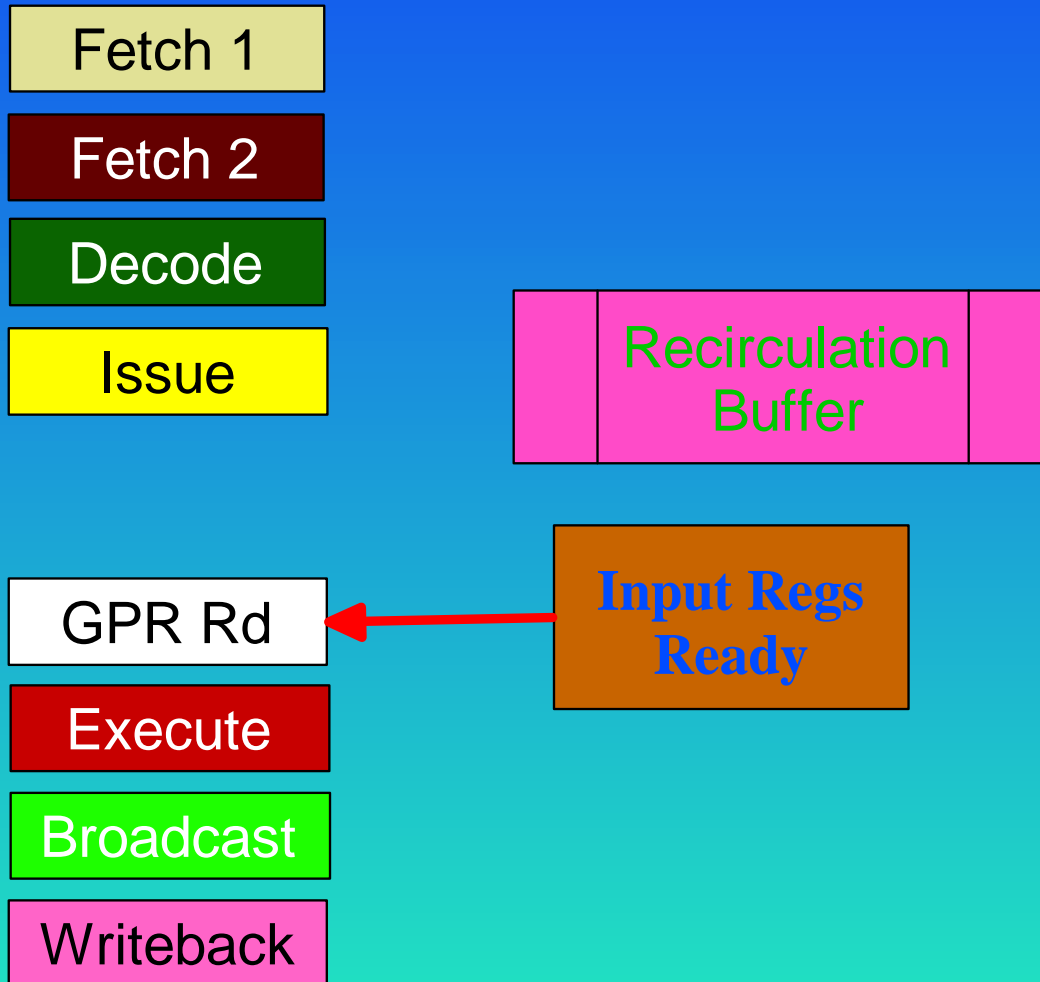
**STORE**

# Recirculation

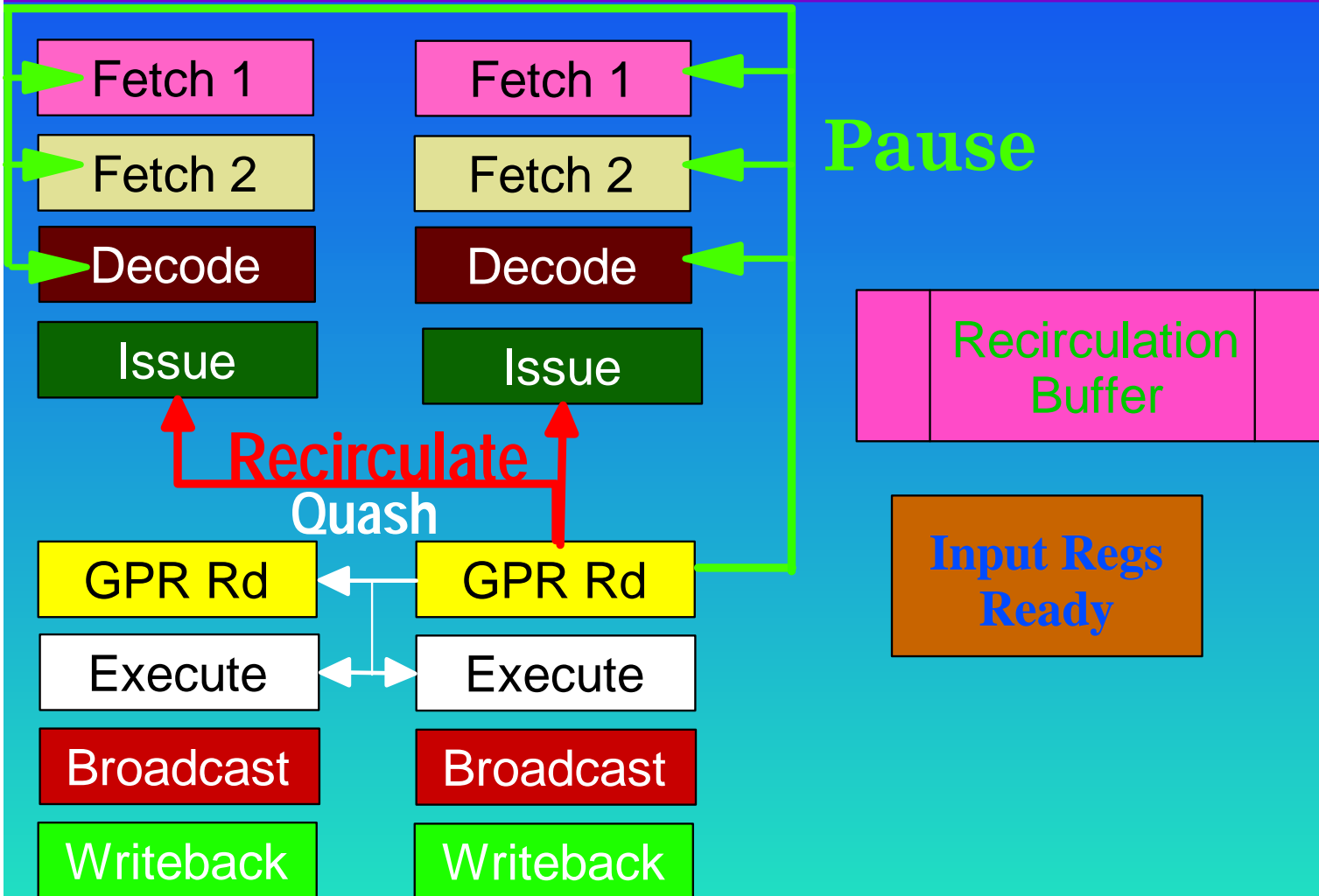


High Frequency =>  
*Not send global stall signals.*  
**Recirculate Ins instead.**

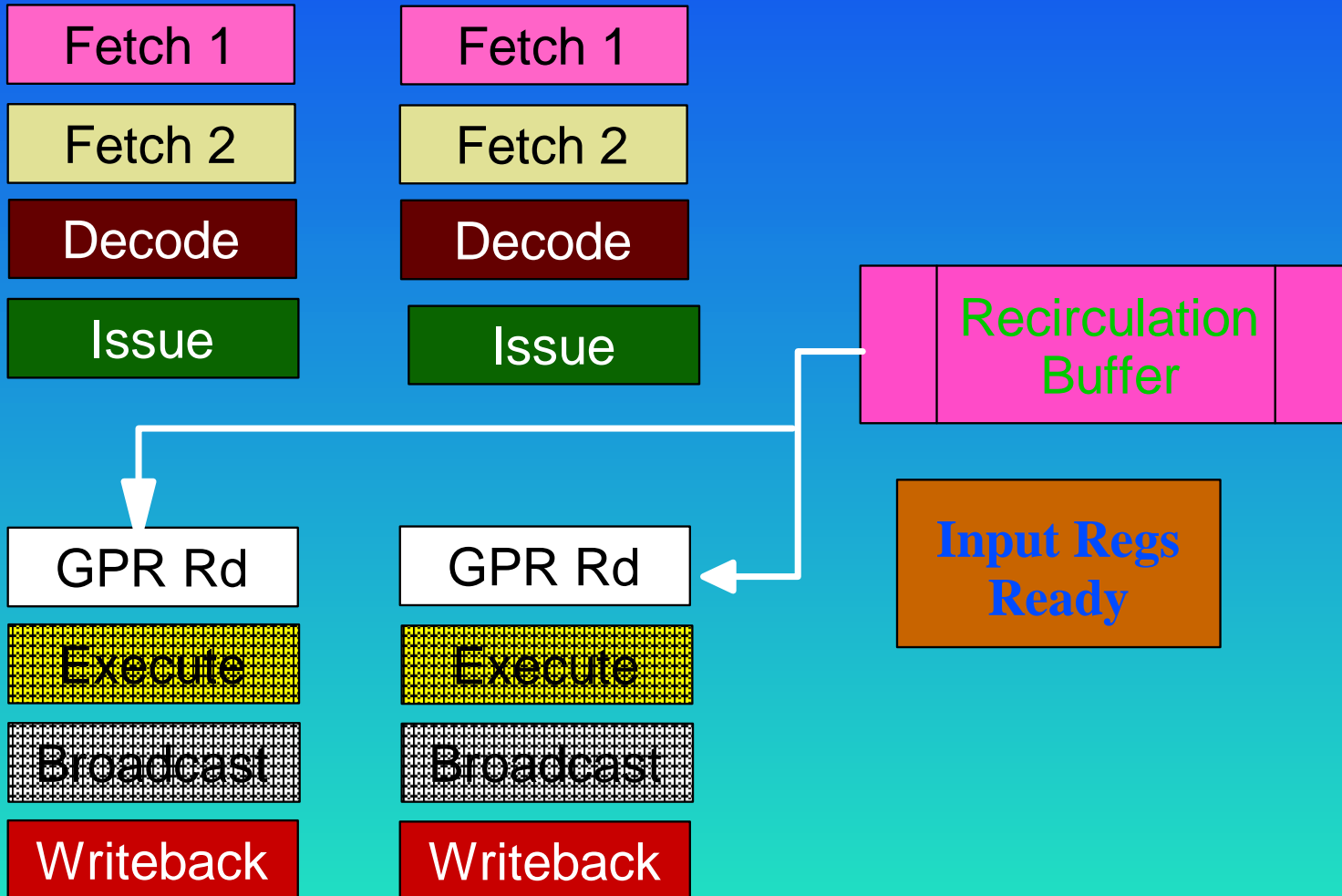
# Recirculation



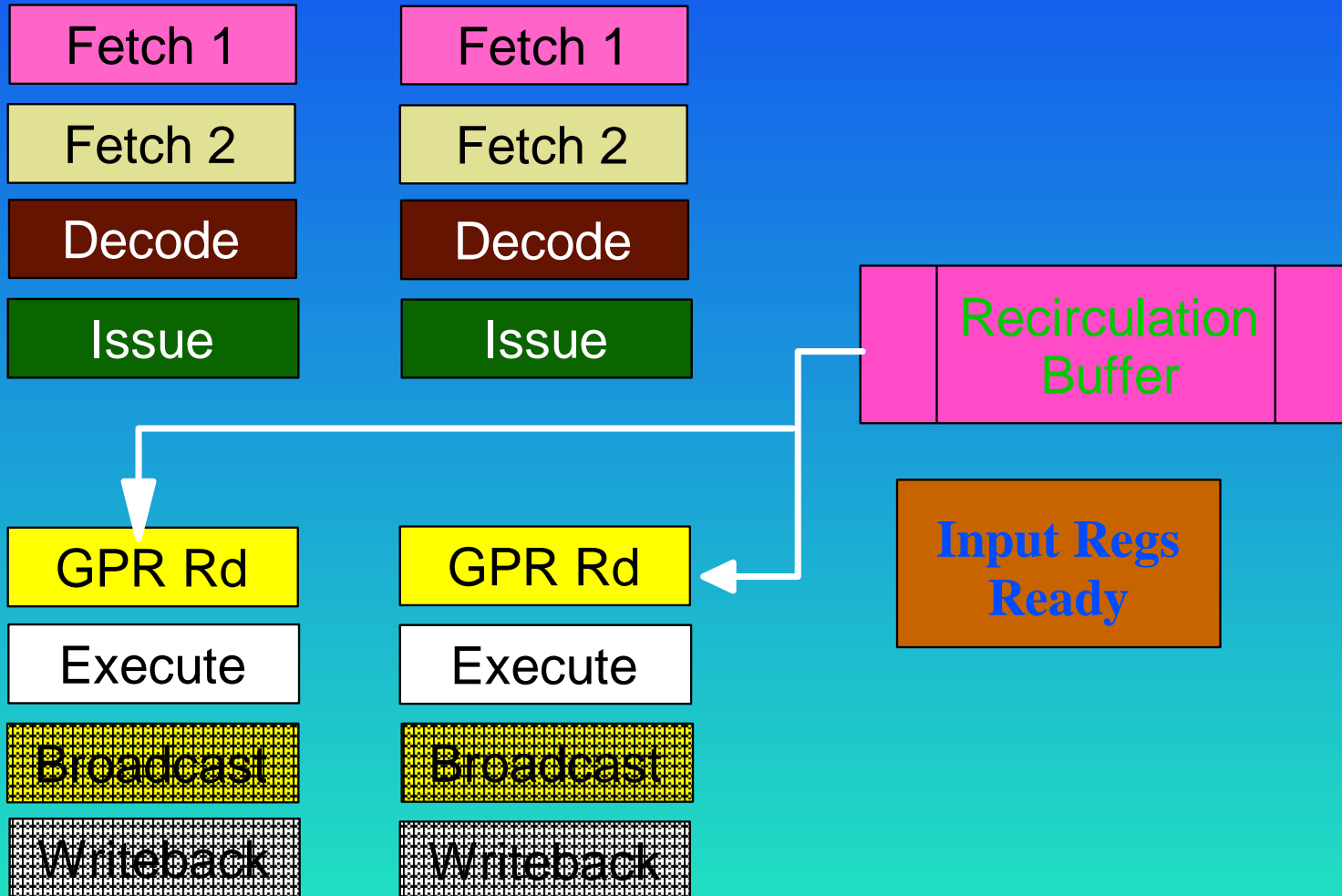
# Recirculation



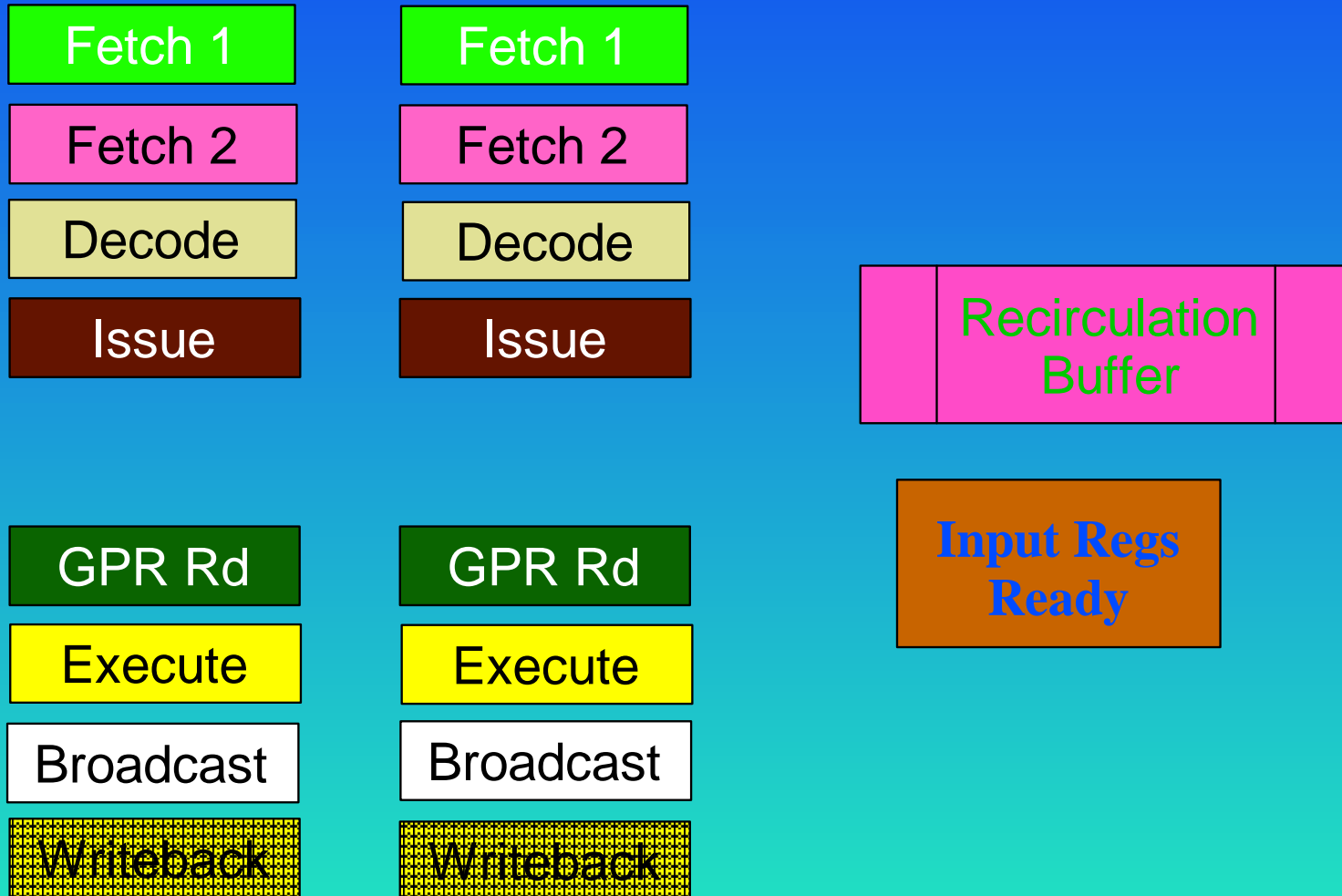
# Recirculation



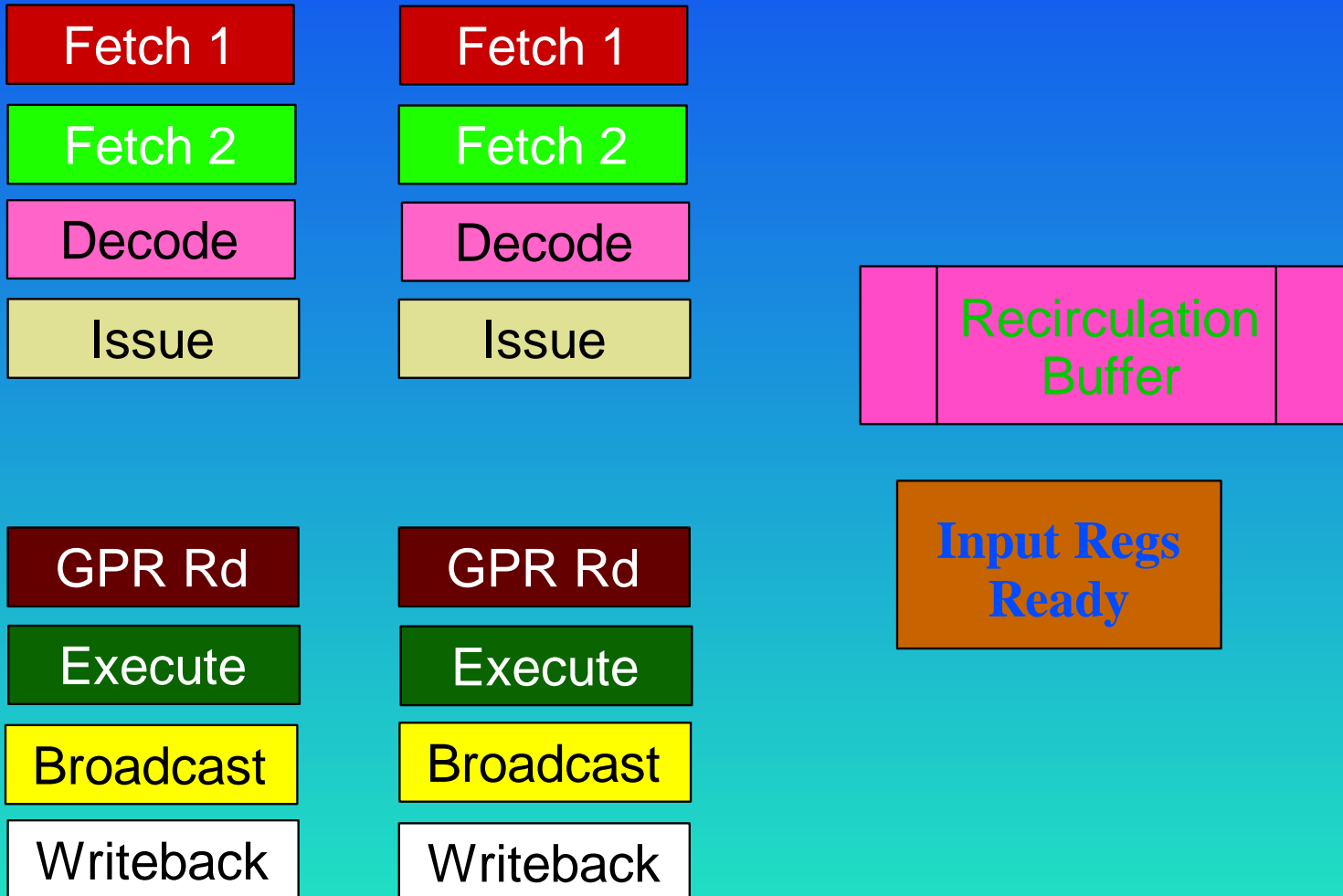
# Recirculation



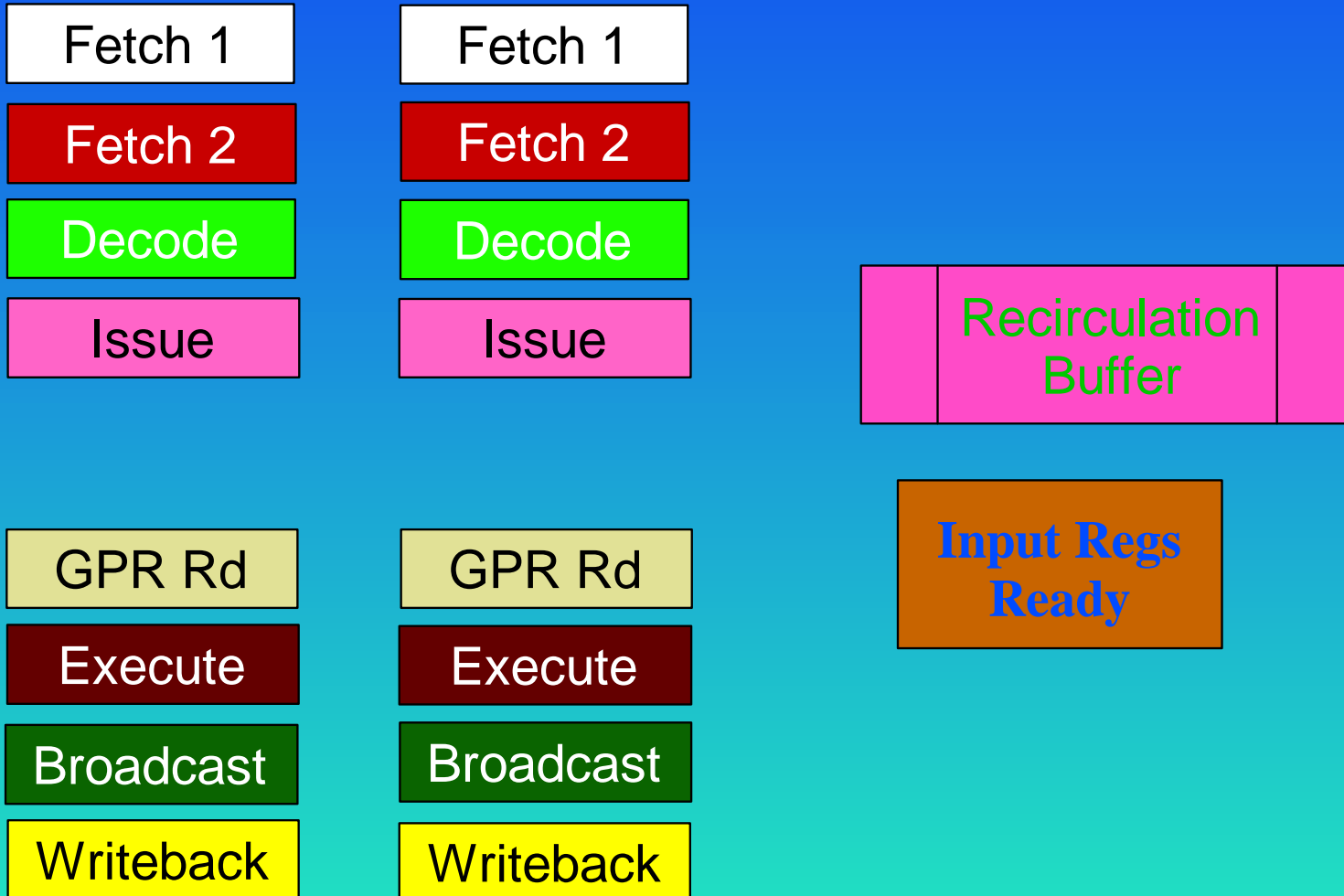
# Recirculation



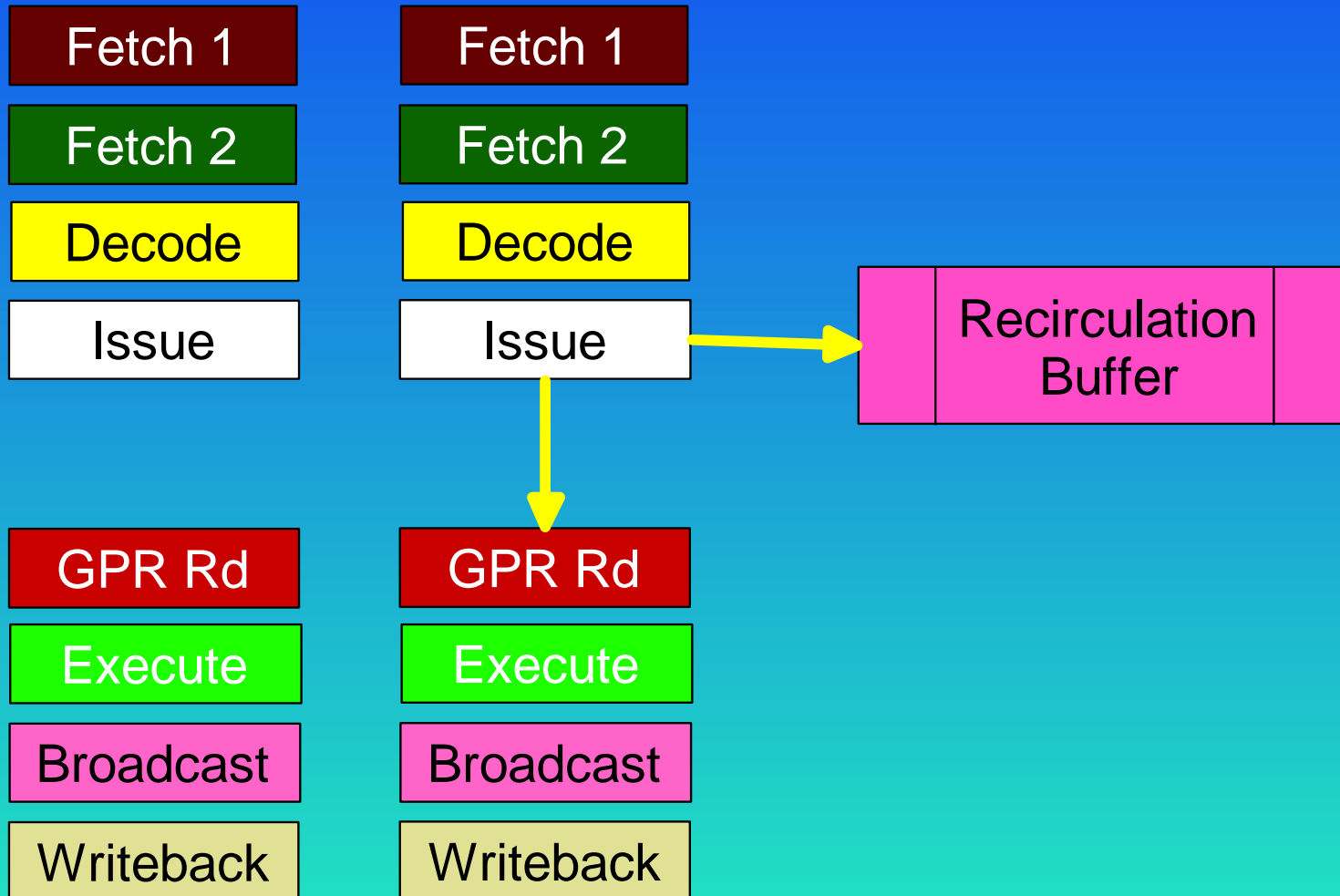
# Recirculation



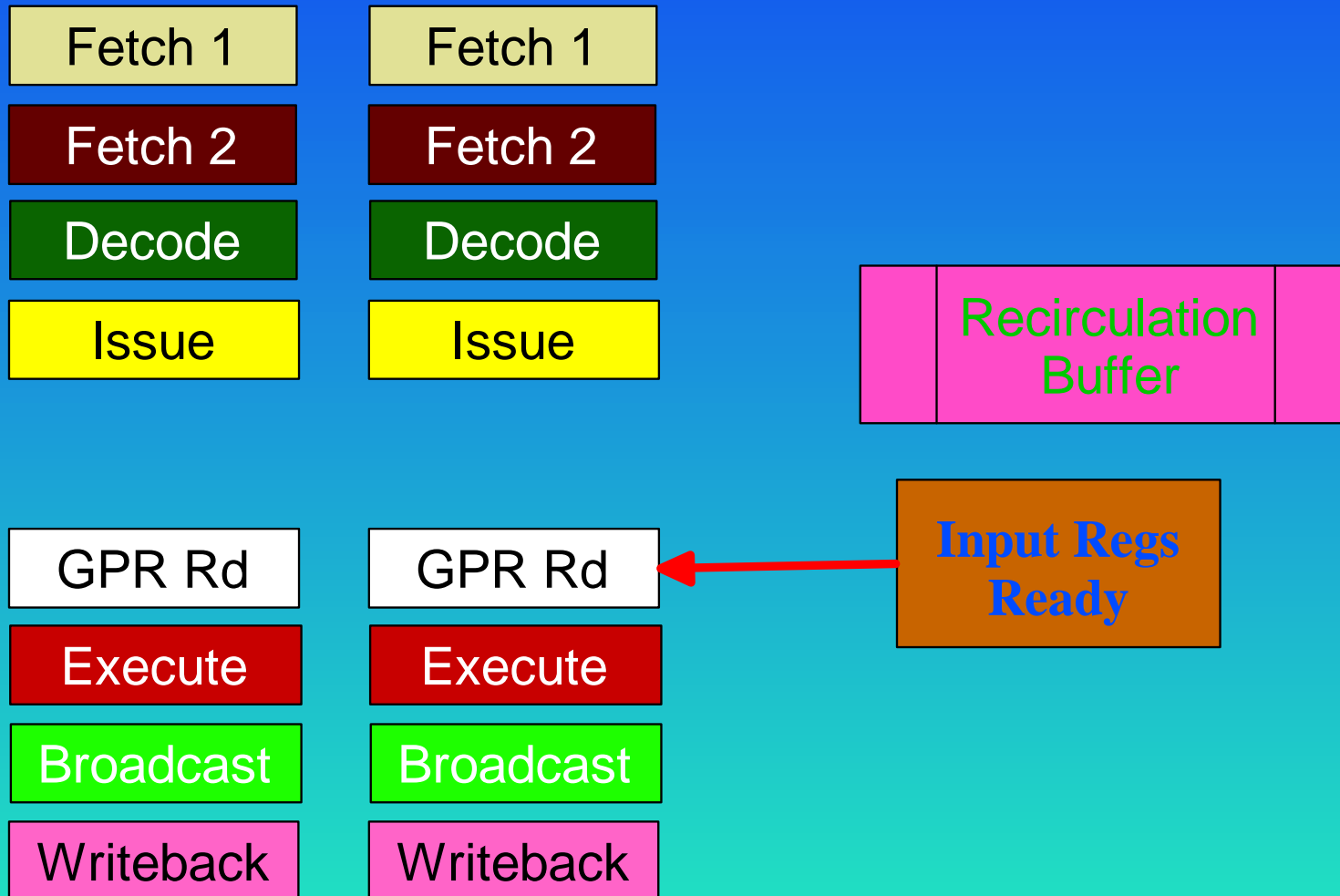
# Recirculation



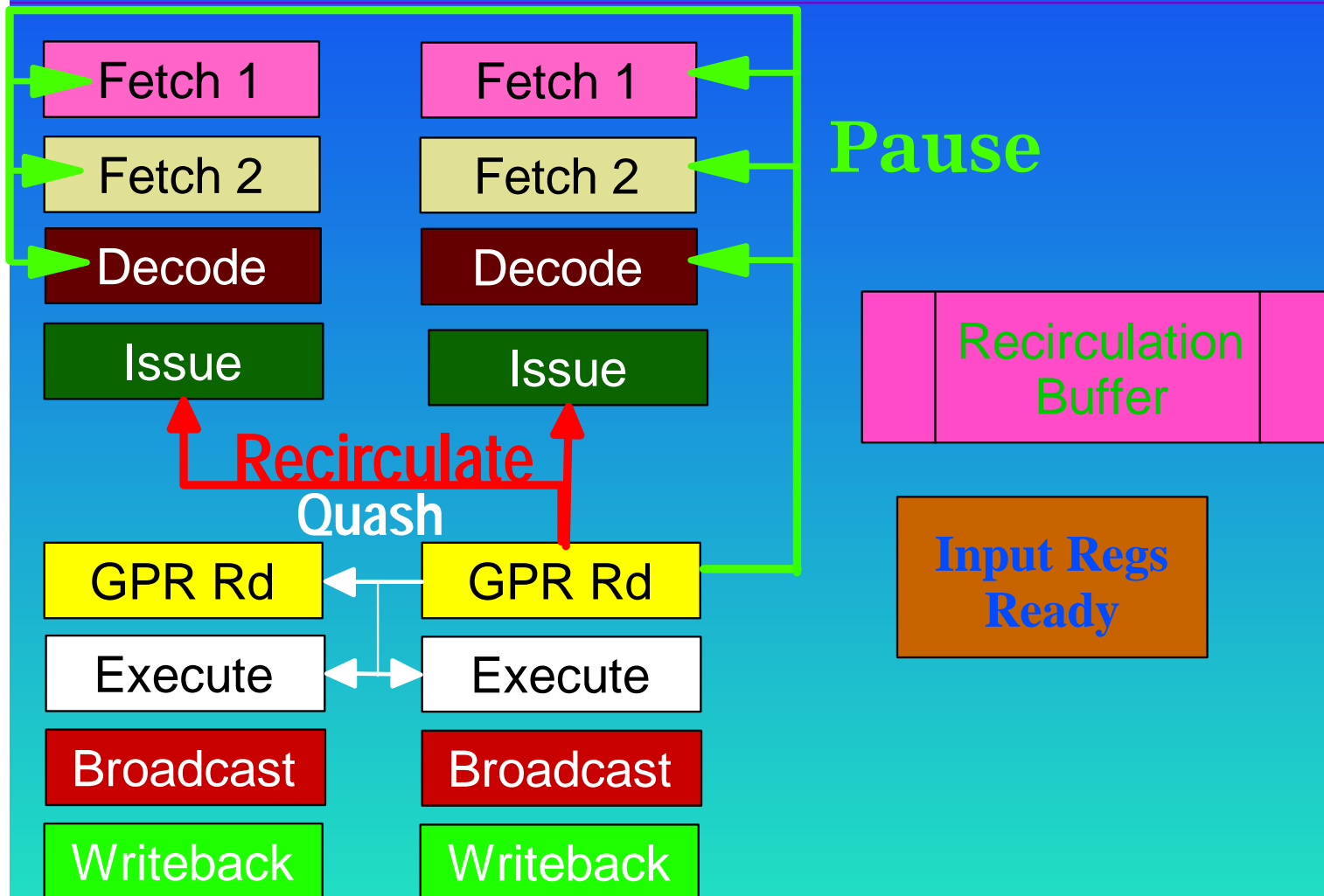
# Recirculation



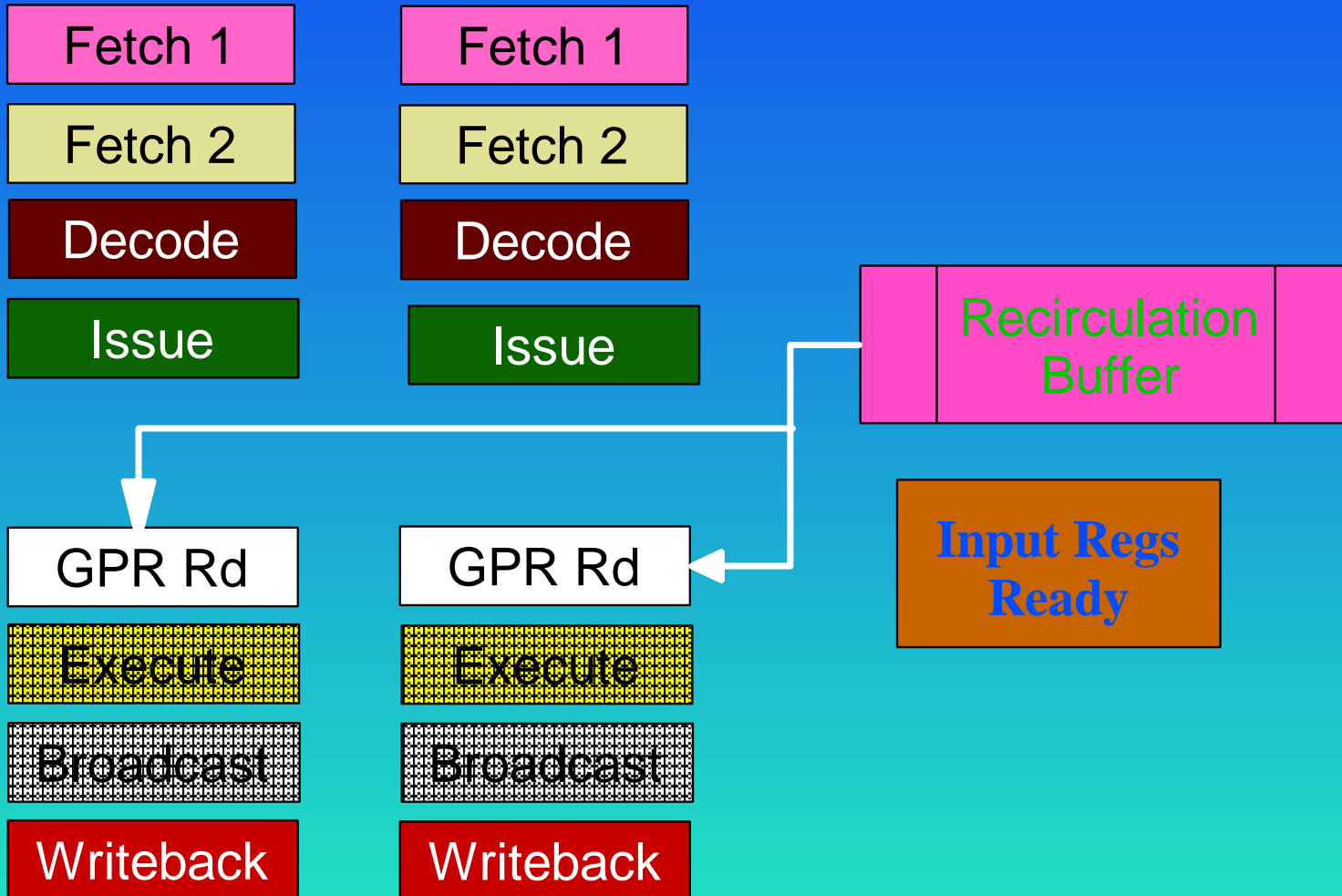
# Recirculation



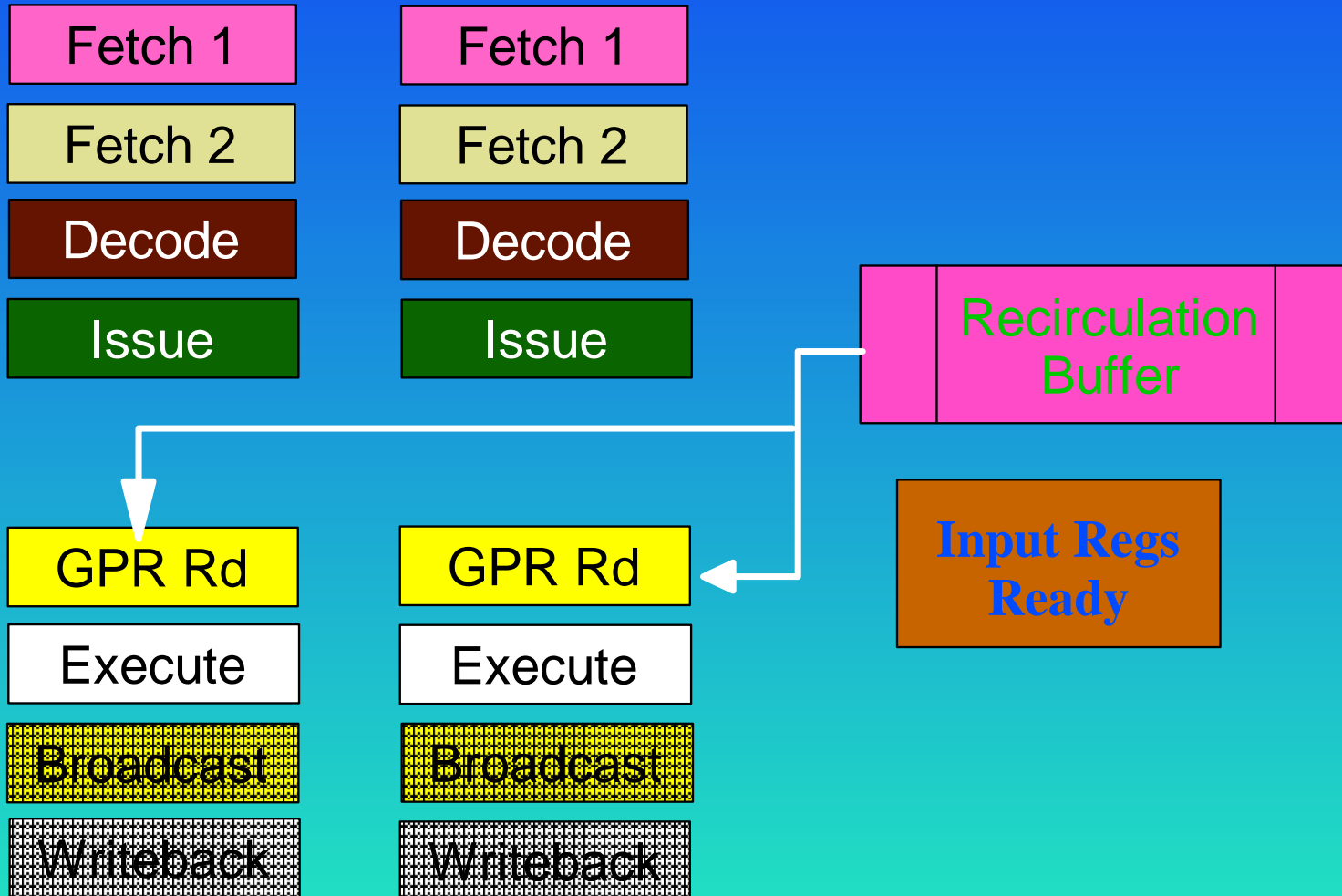
# Recirculation



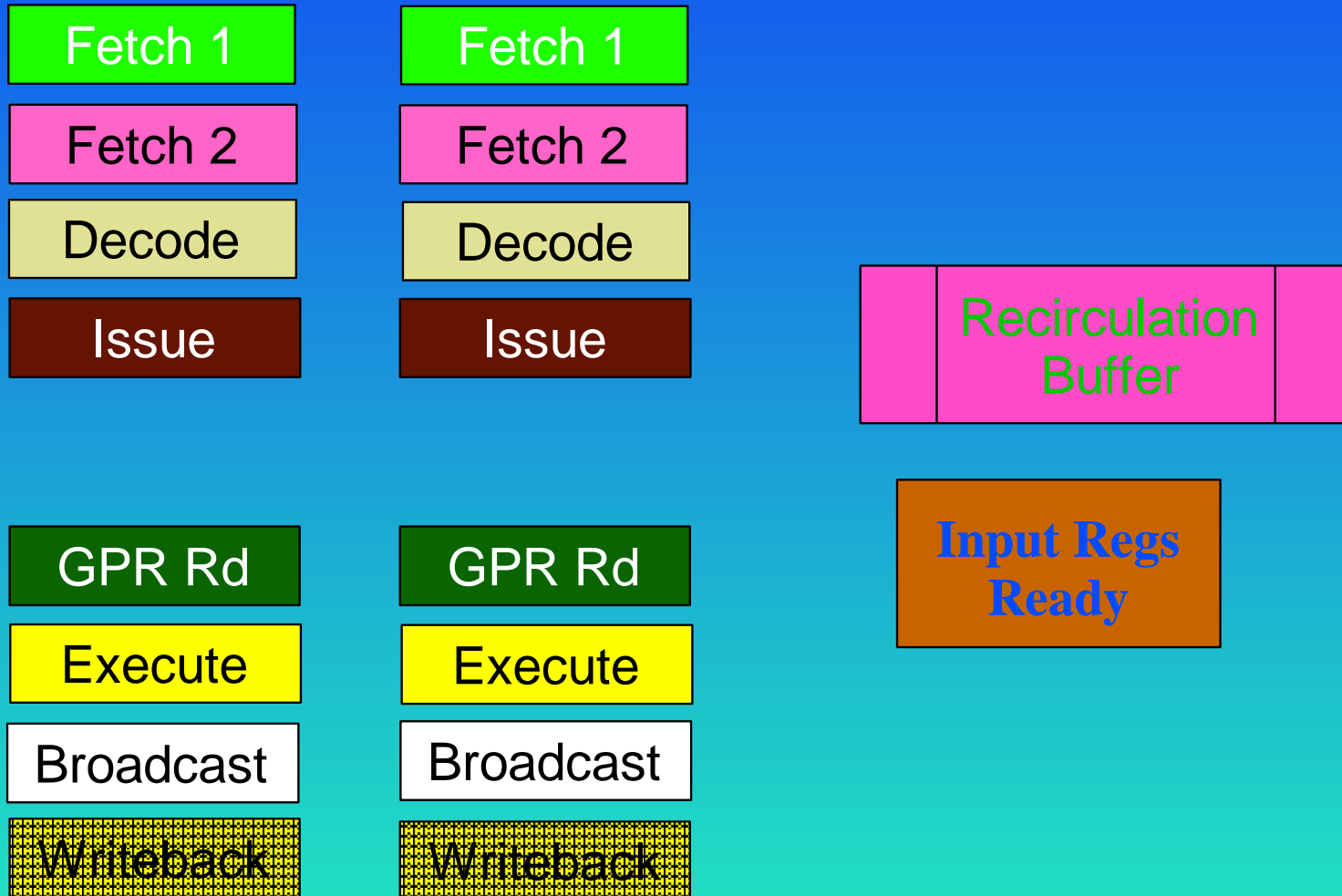
# Recirculation



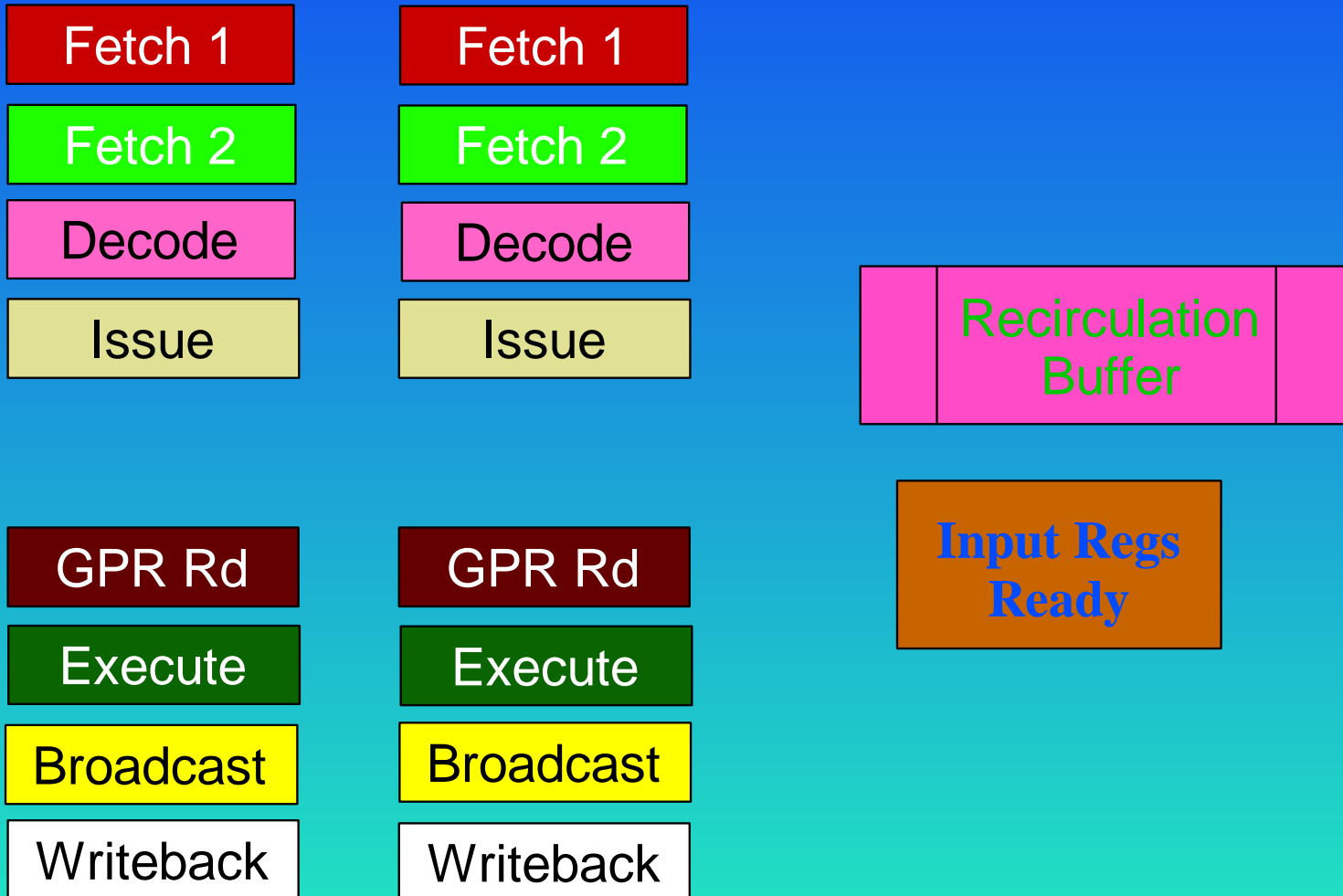
# Recirculation



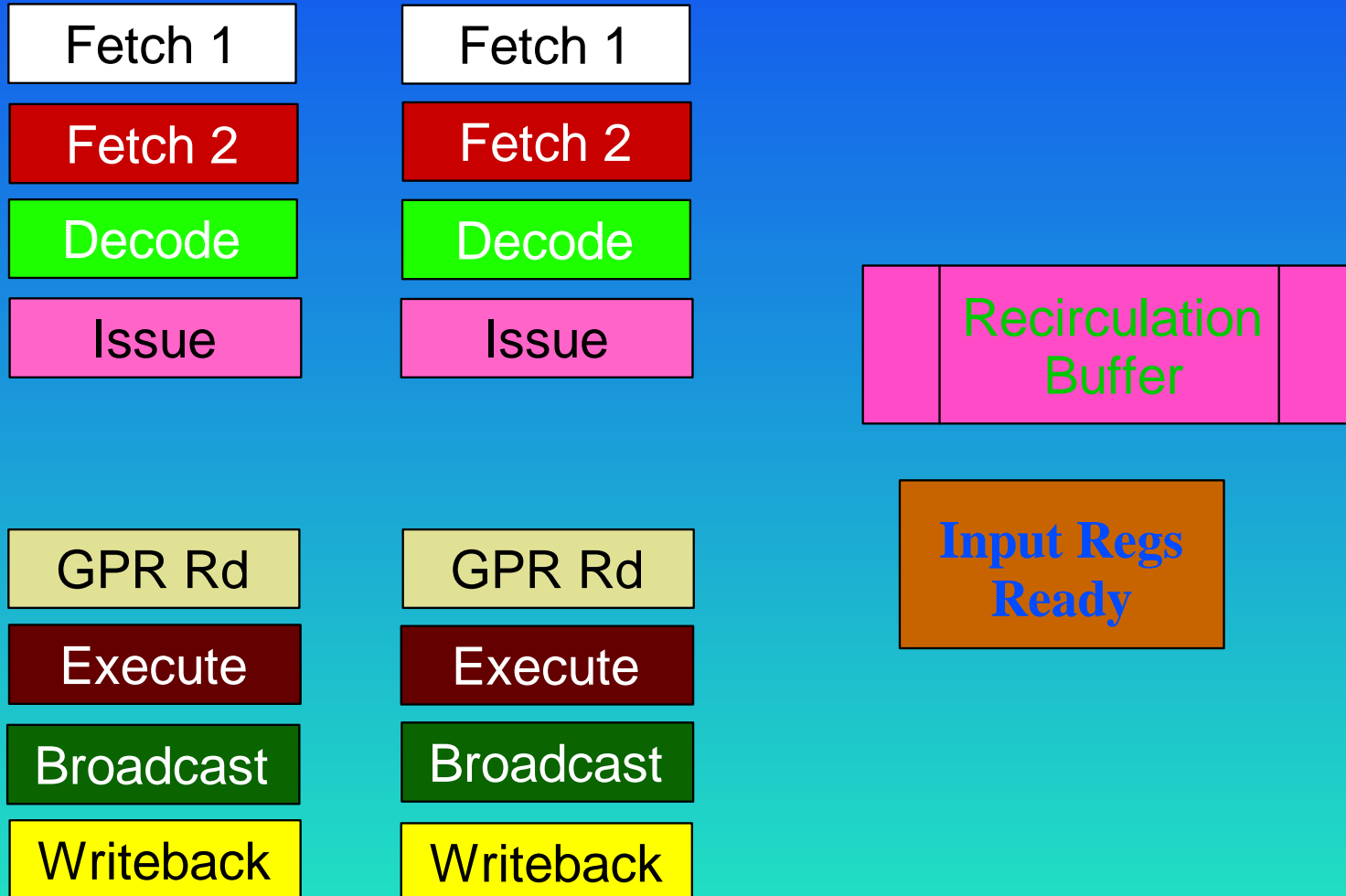
# Recirculation



# Recirculation



# Recirculation



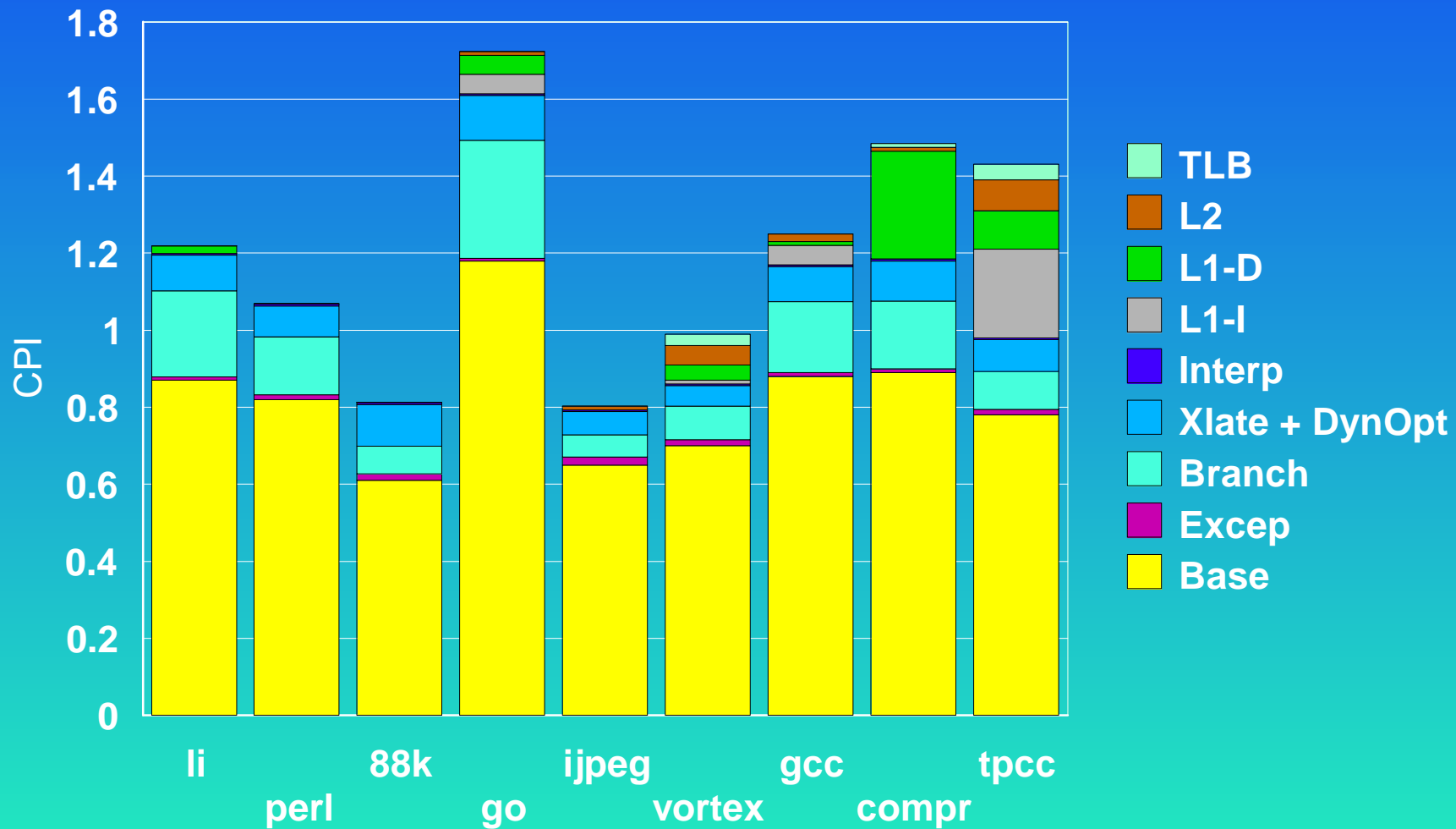
# BOA Caches

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

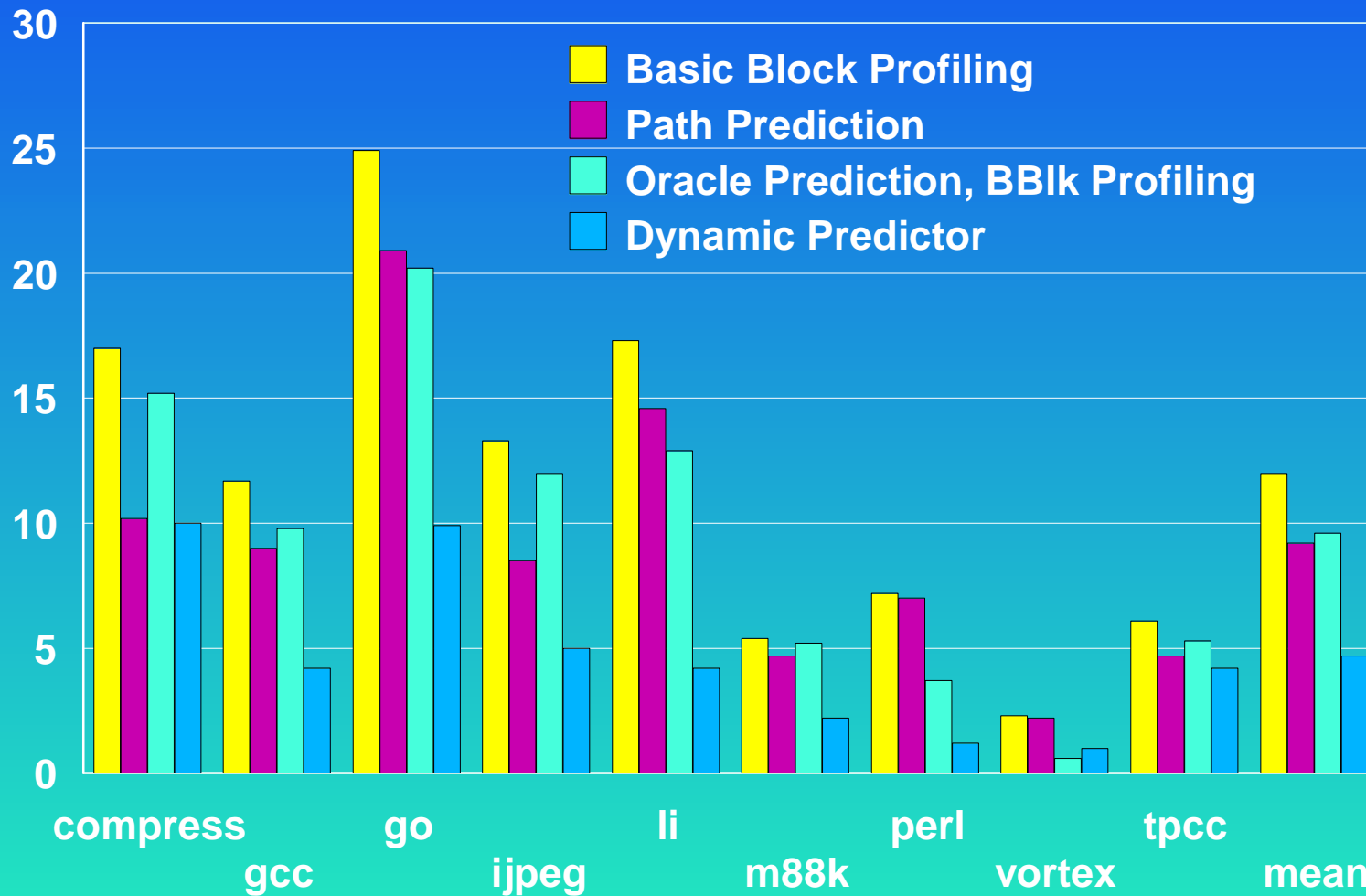
# Benchmarks

- **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

# BOA Baseline CPI



# Branch Misprediction Rates



# **CPI for Block Structured ISA**

# BOA and Other Dynamic Optimization Approaches

- Produces code for a different underlying architecture, unlike *Dynamo*.
- Wider issue, higher frequency, more performance oriented design than *Transmeta*.
- Runs whole architecture and not dependent on OS, unlike *FX!32*.
- Unlike *Java JITS*:
  - ▶ High Level Language Independent
  - ▶ But not portable across platforms

# Conclusions

- **BOA** uses dynamic optimization to simplify processor design.
- **BOA** achieves good performance.

# Conclusions

- **BOA** uses dynamic optimization to simplify processor design.
- **BOA** achieves good performance.
- Cleaned up paper available at:  
[www.research.ibm.com/vliw/Pdf/wced02.pdf](http://www.research.ibm.com/vliw/Pdf/wced02.pdf)  
**or** [www.research.ibm.com/people/m/mikeg/papers/2002\\_wced.pdf](http://www.research.ibm.com/people/m/mikeg/papers/2002_wced.pdf)