Dynamic Binary Translation and Optimization

Erik R. Altman
Kemal Ebcioğlu
IBM T.J. Watson Research Center

Micro-33
December 13, 2000

Timetable for Micro-33 Tutorial on

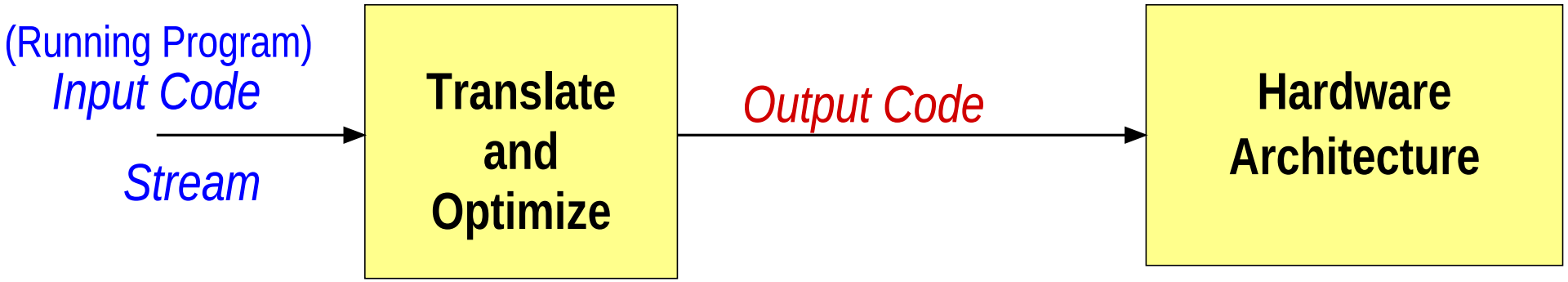
Dynamic Binary Translation and Optimization

Wednesday, December 13, 2000

2:30 - 2:50	Kemal Ebcioglu:	Future Challenges
2:50 - 2:55	Erik Altman:	DAISY Demo
2:55 - 3:20	**Erik Altman**	**Binary Translation Issues**
3:20 - 3:35	Break	
3:35 - 5:00	Erik Altman:	DAISY, Crusoe, Dynamo
5:00 - 5:15	Break	
5:15 - 5:45	Kemal Ebcioglu:	LaTTe

BINARY TRANSLATION ISSUES

Dynamic Binary Translation and Optimization



Note: Translation may be to same or different architecture.

Why do Dynamic Binary Translation and Optimization?

- Improve Performance:
  - Optimization and Profile-Directed Feedback hidden from user.
- Make Architecture a Layer of Software:
  - **Compatibility:** with Legacy Architectures.
  - **Novelty:** use great new architecture ideas.
  - **Multiplicity:** of legacy architectures on one machine.
- Reduce Hardware Complexity.
  - Save Power: Memory less power than logic $\Rightarrow$ Software translator translates once and saves in memory.
  - No need to optimize with logic transistors each time code is executed.

Benefits of Dynamic Optimization (1)

- Legacy code where source is unavailable can be optimized.
- Many commercial programs are never compiled with **-O** due to time-constraints on shipping product.
  - Development done with **-g**, not **-O**.
  - No time for additional testing with **-O** compilation.
- Dynamic Optimization still allows high code quality.

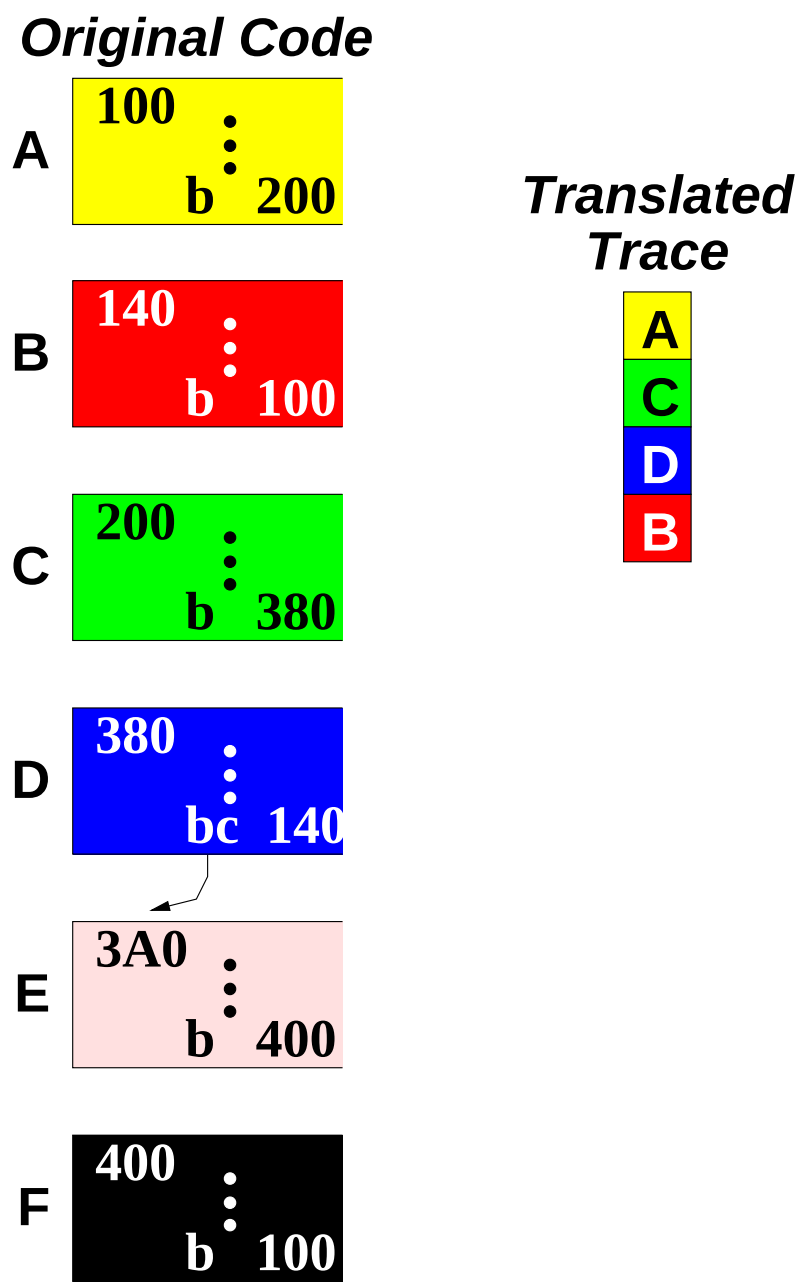
Benefits of Dynamic Optimization (2)

- Not limited in optimization scope. Can cross boundaries:
  - Indirect Calls
  - Function Returns
  - Shared Libraries
  - System Calls

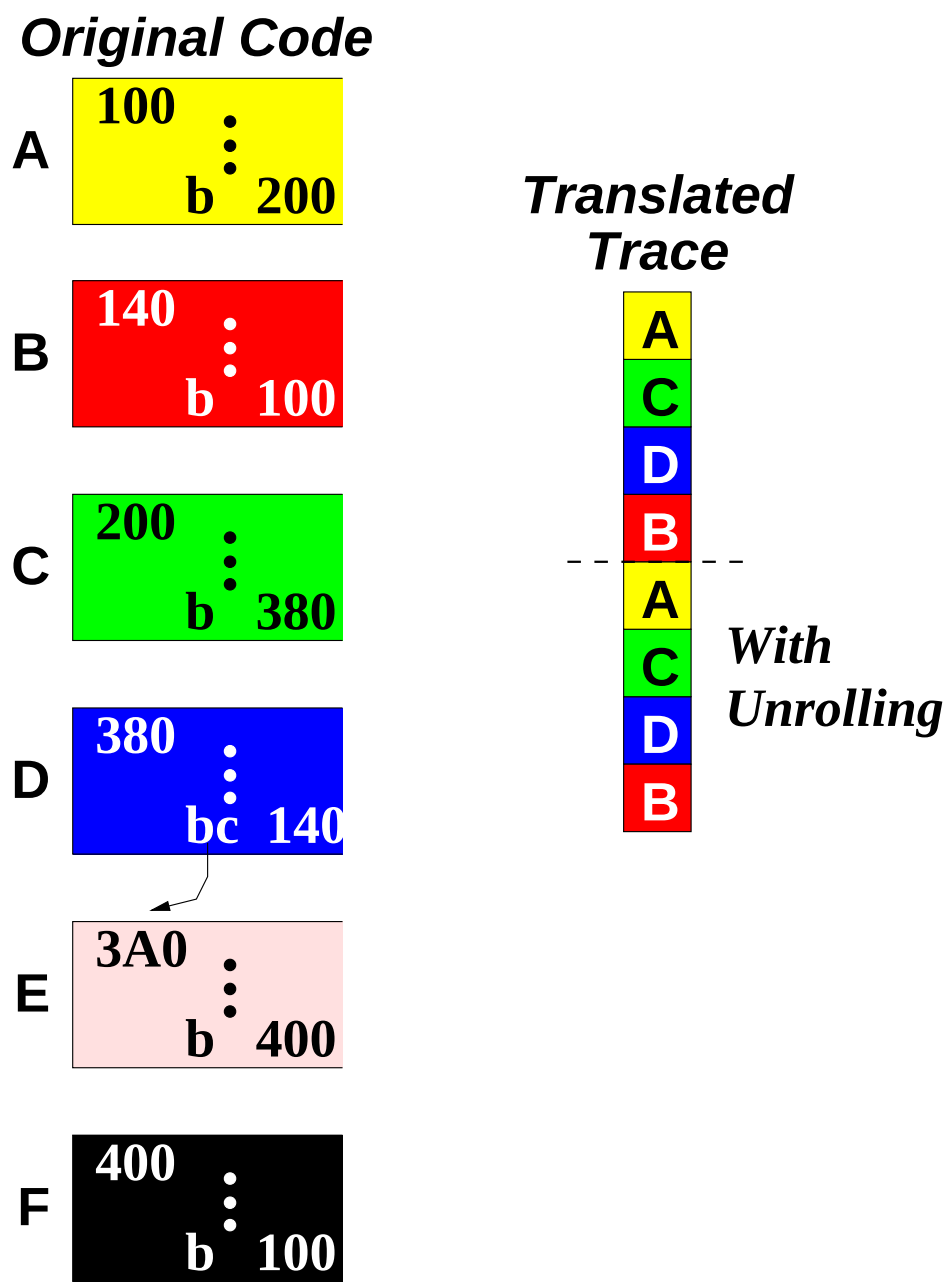
Benefits of Dynamic Optimization (3)

- Translated basic blocks can be layed out contiguously in order they are naturally visited.
- This helps ICache Performance.
- **Example:** . . .

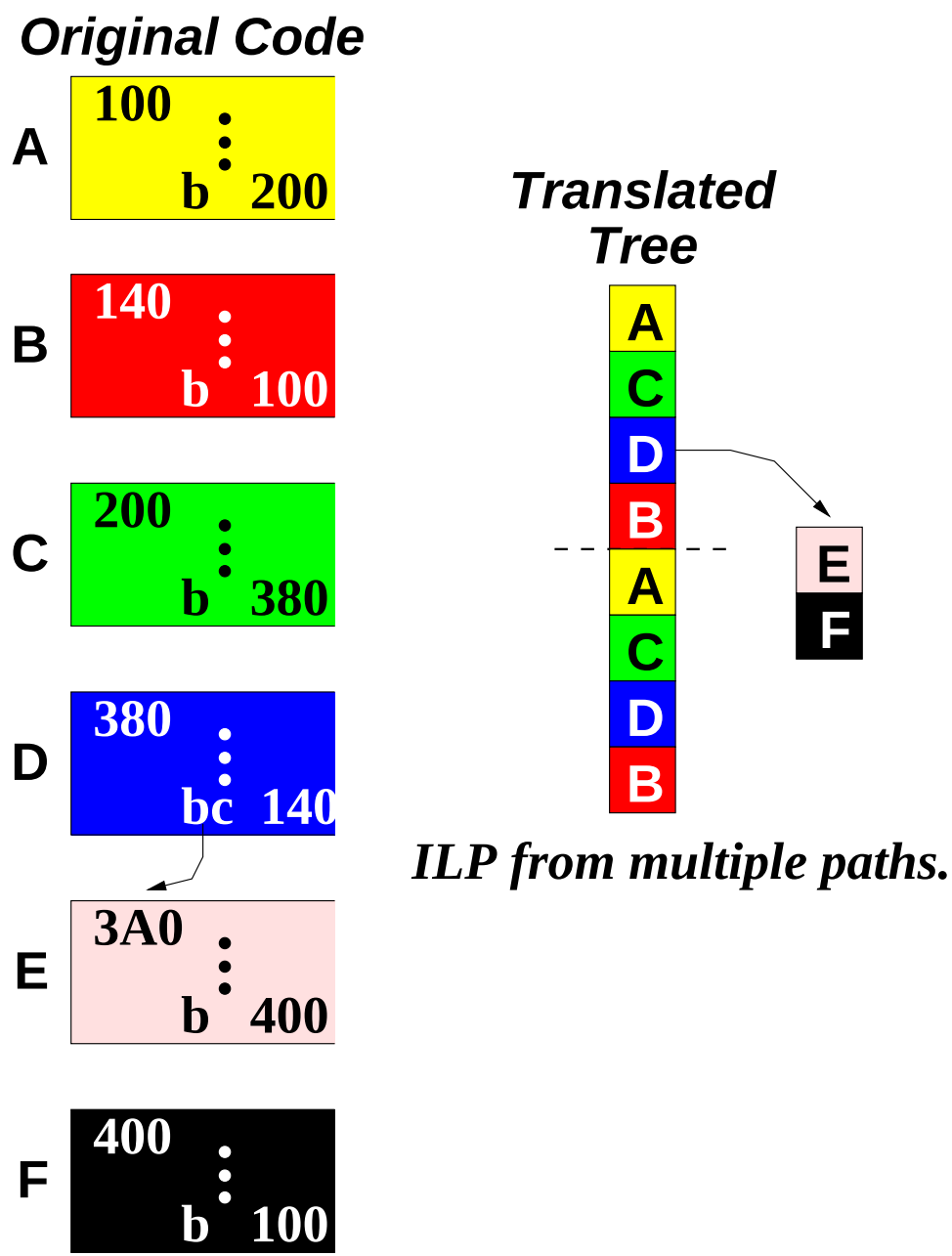
Example of ICache Layout Benefit



Example of ICache Layout Benefit



Example of ICache Layout Benefit



ILP from multiple paths.

Benefits of Dynamic Translation (1)

- Compatible: No changes are needed in existing code.
- Compatible: between VLIW's of different sizes, generations.
  - *Transmeta already uses this advantage.*
- Upgradable: If better compiler algorithms are found, only a software patch is needed to install them.
- Upgradable: Future architectural improvements are transparent to the user.

Benefits of Dynamic Translation (2)

- Reliable: If a bug is found in dynamic translator a software patch is sufficient to fix it.
- Reliable: Some hardware bugs can be worked around by translator.
- High Chip Yield: Smarts in software not hardware $\Rightarrow$ Smaller chip with higher yield.
- Simple Fast Hardware: Intelligence is in software, allowing simple in-order implementations.
- Wide scope for ILP: Can look at arbitrarily long fragments of code.

Problems with Dynamic Translation

- Takes away cycles from program.
- Takes memory and resources from emulated machine.
- Especially slow at start.
- Potential realtime difficulties.
- Debugging can be difficult:
  - Target machine code several times removed from source code.
  - Behavior can be non-deterministic in real system.

Problems with Static Translation

- Finding all code.
- Self-modifying code problematic:
  - May require considerable additional hardware, e.g. a "spare tire" — hardware for the **source architecture** in addition to the **target architecture**.
- Support of shared library and system calls.

History of Binary Translation

- **1987:** HP ships binary translation system from the HP-3000 to PA-RISC.
- **1991:** Tandem ships interpreter and translator system.
- **1992:** DEC ships set of binary translators to *Alpha* architecture from VAX/VMS, MIPS/Unix, Sparc/Unix, and x86/NT.
- **1994:** Apple ships Motorola 68000 interpreter. WABI, VirtualPC and other programs to run Windows programs on other platforms.
- **1996:** Sun releases *Java* JDK 1.0.
- **1996:** IBM Research starts **DAISY** project.
- **2000:** Transmeta ships **Crusoe** processor.

From Dick Sites, *"A Few Notable Events in the History of Binary Translation"*, March 2000, IEEE Computer

Selected Binary Translation Projects

- IBM **DAISY**
- IBM BOA
- Transmeta **Crusoe**
- HP **Dynamo**
- HP Aries
- DEC / Compaq FX!32
- **LaTTe** and Java JIT's
- Tandem CISC $\Rightarrow$ RISC
- WABI
- UQBT
- VMWare
- Virtual PC
- MIMIC
- Fundamental Software FLEX-ES
- Harvard Deco Project
- Pentium Pro Hardware Cracking
- Shade
- DyC
- Many more at: **http://www.xsim.com/bib/index2.d/Todo-9.html**

Focus of this Tutorial

- **Part I** of Tutorial emphasizes binary translation approaches that are comparable to or better than a native implementation:
  - **IBM DAISY**.
  - **Transmeta Crusoe**.
  - **HP Dynamo**.
- **Part II** of Tutorial examines emulation of virtual machines, e.g. Java:
  - **LaTTe**

Emulator Performance

- **DAISY** achieves ILP up to 3-4 instructions per cycle.
- "The performance of the **[Transmeta] T5400** running at 667 MHz is about the same as a Pentium III running at 500 MHz" – *Doug Laird, Transmeta Vice President of Product Development.* See

  http://cnet.com/news/0-1003-200-1526340.html?tag=st.ne.ni.rnbot.rn.ni
- **Dynamo** improves execution time by up to 22%.
- **LaTTe** has performance 8% to 30% better than Sun JVM's.

DISCLAIMER

A few diagrams, charts, and other material in this presentation about HP's **Dynamo** and Transmeta's **Crusoe** are based on published material about these two projects.

BACKGROUND

Nomenclature

- Source Architecture = Machine *from* which translating.
- Target Architecture = Machine *to* which translating.
- VMM = Virtual Machine Monitor, the sub-operating system software controlling binary translation and other low level functions.
- TCache = Translation Cache, the memory where translations are stored. The TCache is not necessarily a hardware cache.

Difficult Problems

- Self Modifying Code
- Precise Exceptions
- Address Translation
- Self Referential Code
- Management of Translations
- Realtime Code
- Boot code

Tradeoffs: Binary Translation Space

- Static *vs* Dynamic.
  - **Static** requires new software tools for binary modules, and is usually not 100% compatible with old software.
- Interpret *vs* Translate/Optimize.
- Emulate **virtual** machine *vs* Emulate **real** machine.
- Full System *vs* User.
- OS Independent *vs* OS Dependent.
- Translate to Different Arch *vs* Translate to Same Arch.
- Emulate single source architecture *vs* Emulate multiple.

Tradeoffs: Binary Translation Space

- Static *vs* Dynamic.
  - **DAISY**, **Crusoe**, **Dynamo** and **LaTTe** are all dynamic.

Tradeoffs: Binary Translation Space

- Interpret *vs* Translate/Optimize.
  - **DAISY**, **Crusoe**, **Dynamo**, and **LaTTe** do all of these.
  - **Dynamo** translates to the same instruction set.
  - **DAISY**, **Crusoe**, **Dynamo** and **LaTTe** do as little interpretation as possible.

Tradeoffs: Binary Translation Space

- Emulate **virtual** machine *vs* Emulate **real** machine.
  - **LaTTe** emulates a virtual machine.
  - **DAISY**, **Crusoe** and **Dynamo** emulate real machines.

Tradeoffs: Binary Translation Space

- Full System *vs* User.
  - **DAISY**, **Crusoe**, and **LaTTe** emulate full systems.
  - **Dynamo** emulates user level only.

Tradeoffs: Binary Translation Space

- OS Independent *vs* OS Dependent.
  - **DAISY** and **Crusoe** are OS Independent.
  - **Dynamo** and **LaTTe** are OS Dependent.

Tradeoffs: Binary Translation Space

- Translate to Different Arch *vs* Translate to Same Arch.
  - **DAISY**, **Crusoe** and **LaTTe** translate to a different architecture.
  - **Dynamo** translates to the same architecture.

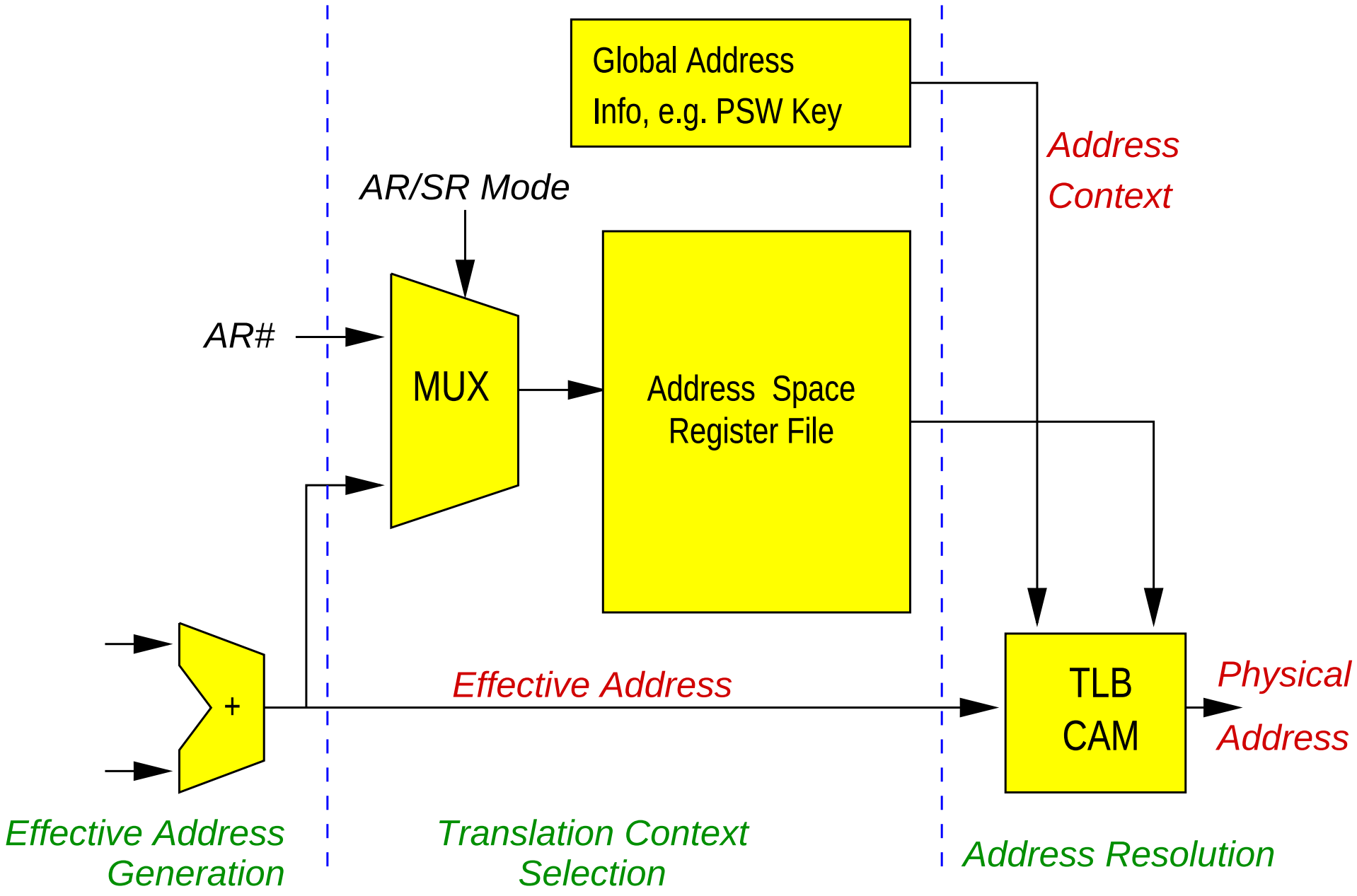
Tradeoffs: Binary Translation Space

- Emulate single source architecture *vs* Emulate multiple.
  - **Crusoe**, **Dynamo** and **LaTTe** all emulate single source architecture.
  - **DAISY** emphasizes **PowerPC**, but some work has been done to emulate **S/390**.

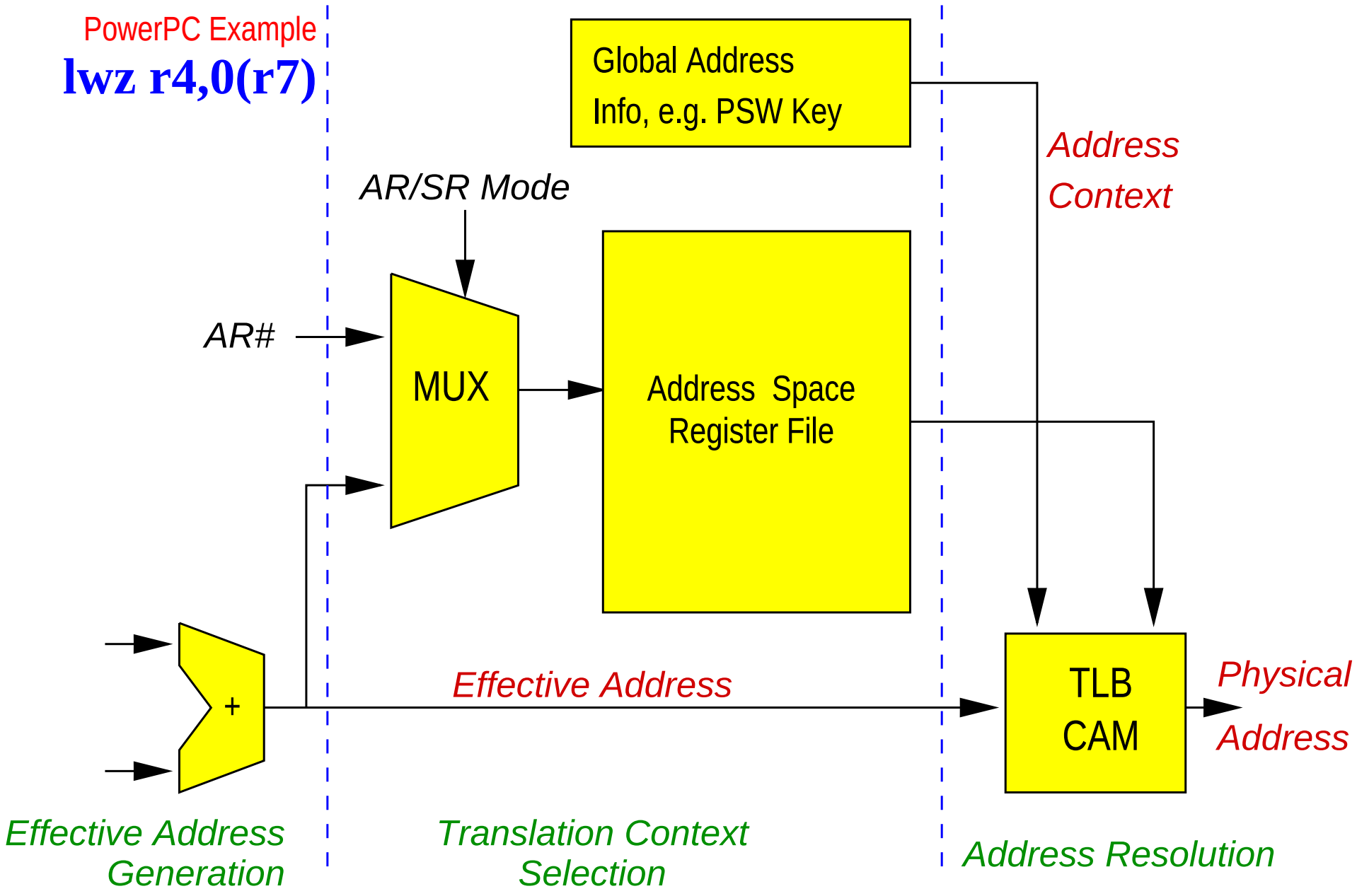
Commonality: Emulating Multiple Legacy Architectures

- **Target architecture** should have hardware support for frequently used features of each legacy architecture.
  - Opcodes.
  - Condition code registers.
  - Floating point formats.
  - Timer registers.
  - Segment registers.
  - Address translation.

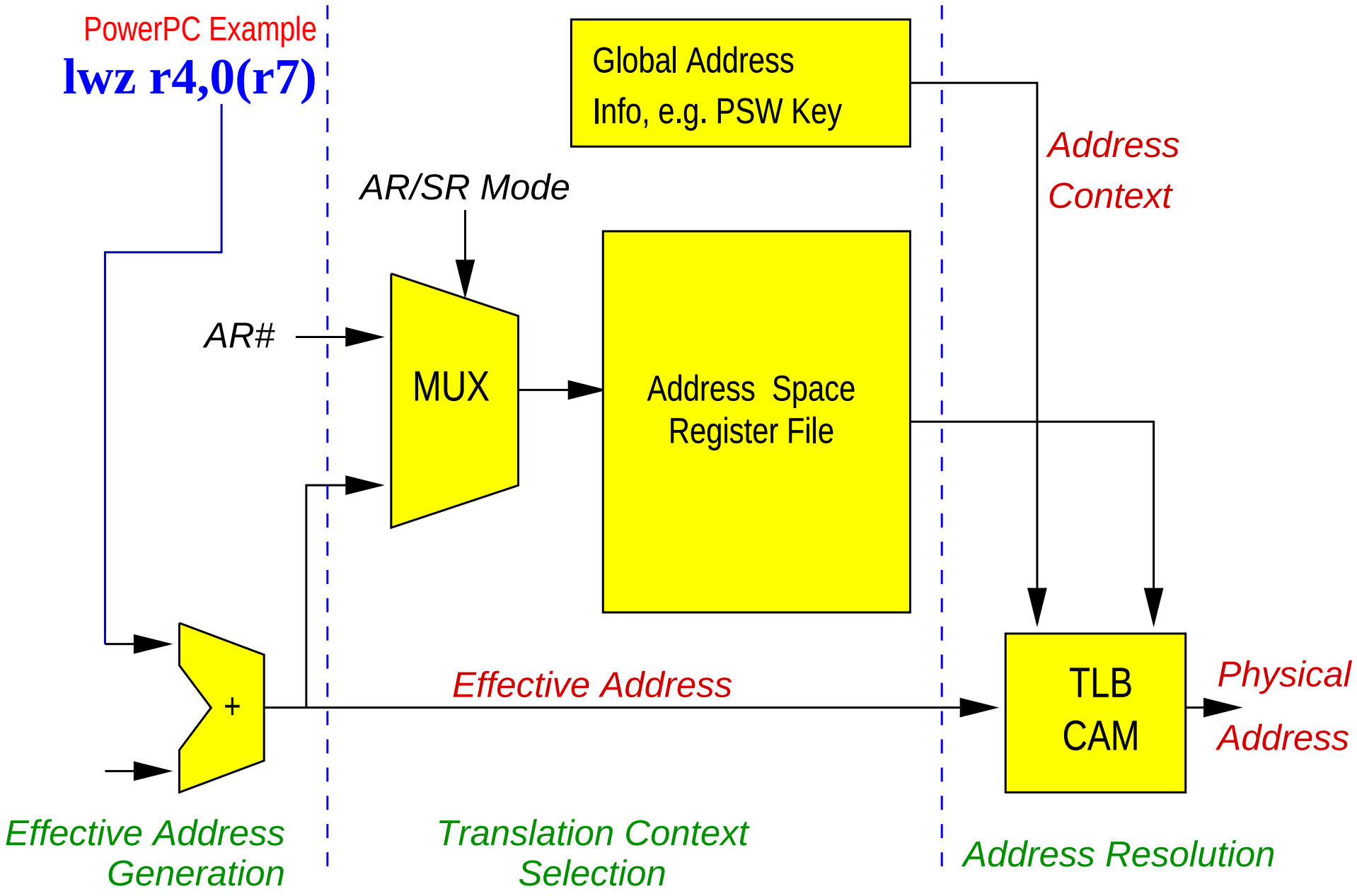
Address Translation Commonality



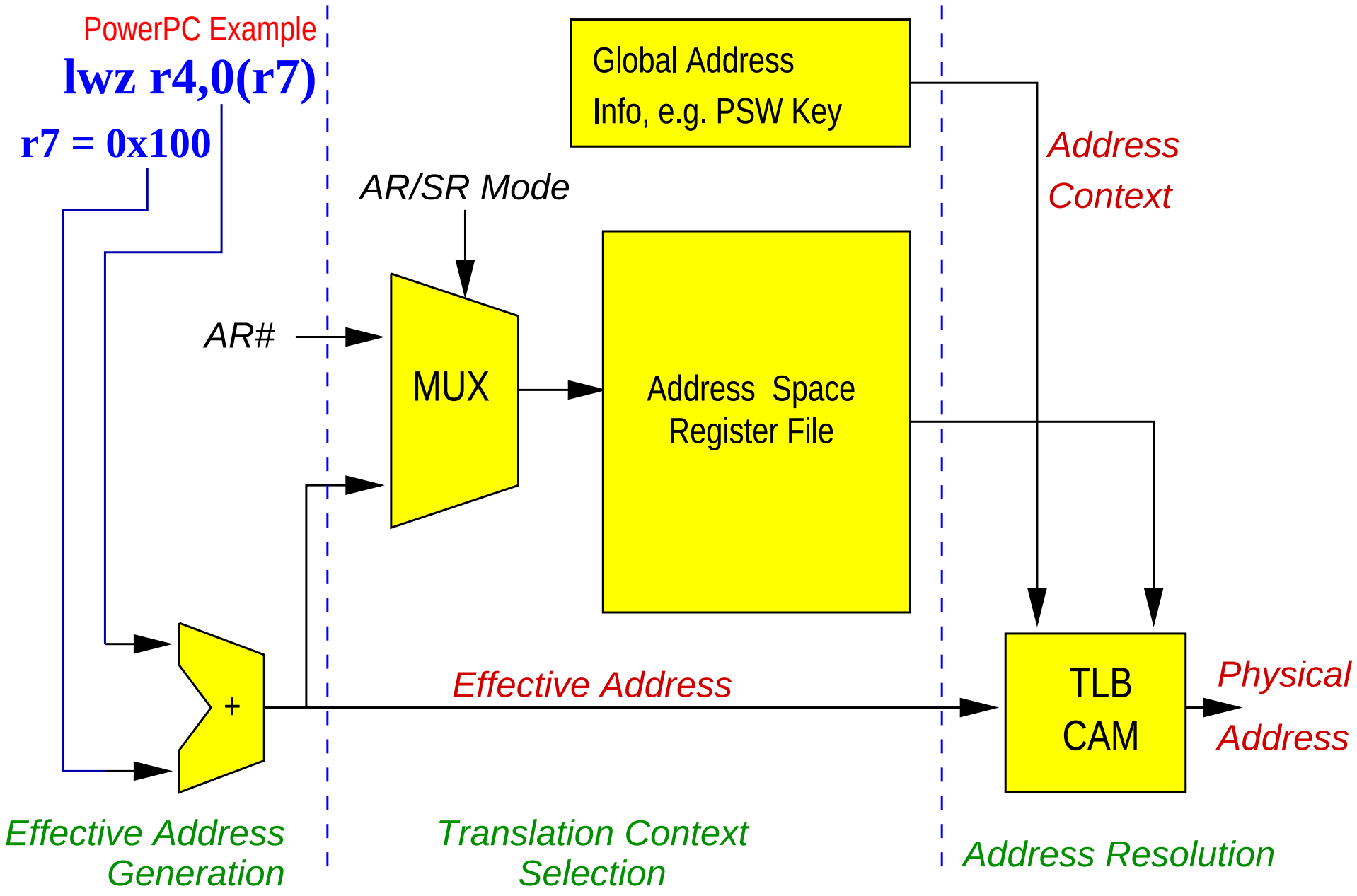
Address Translation Commonality



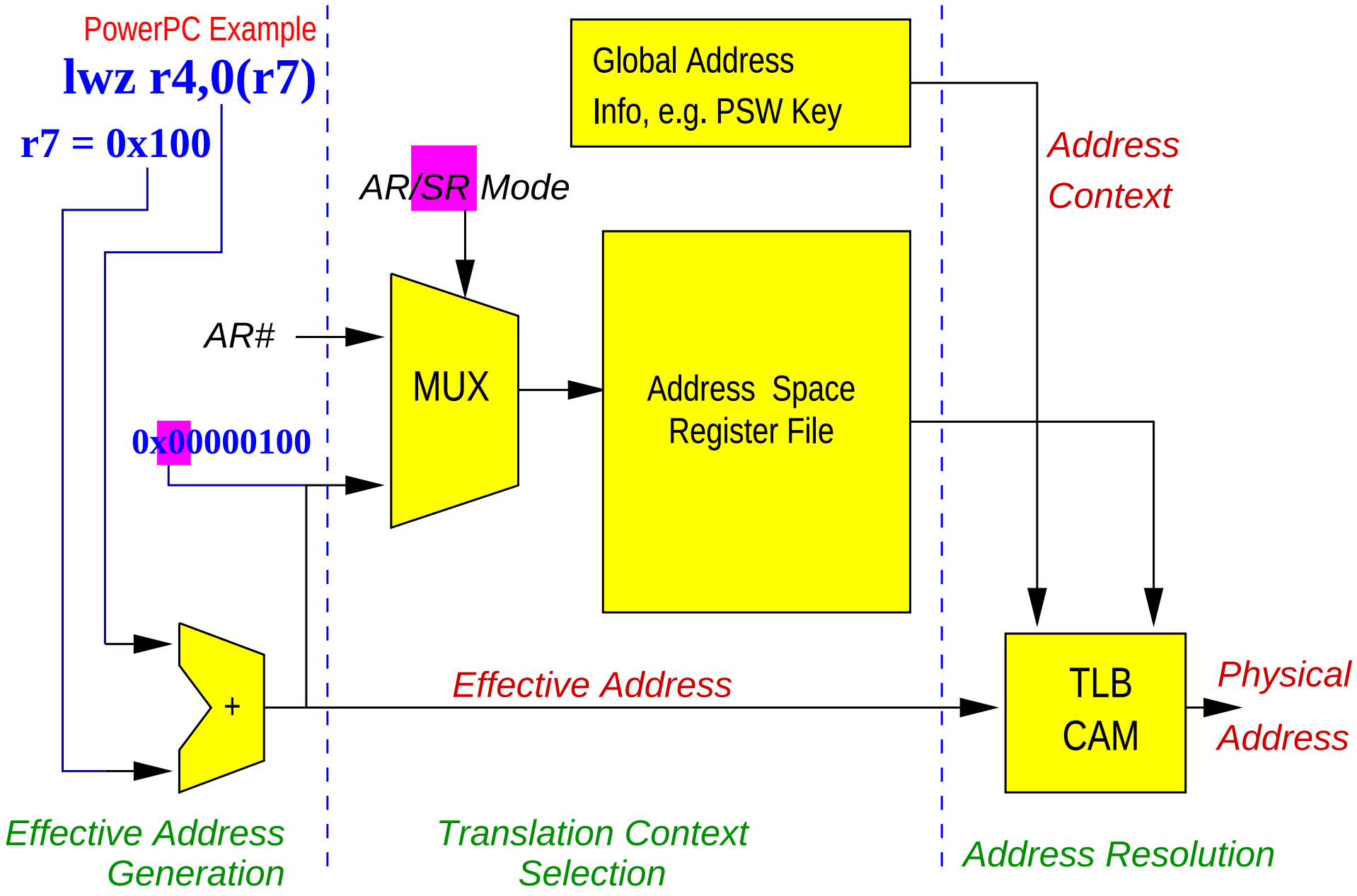
Address Translation Commonality



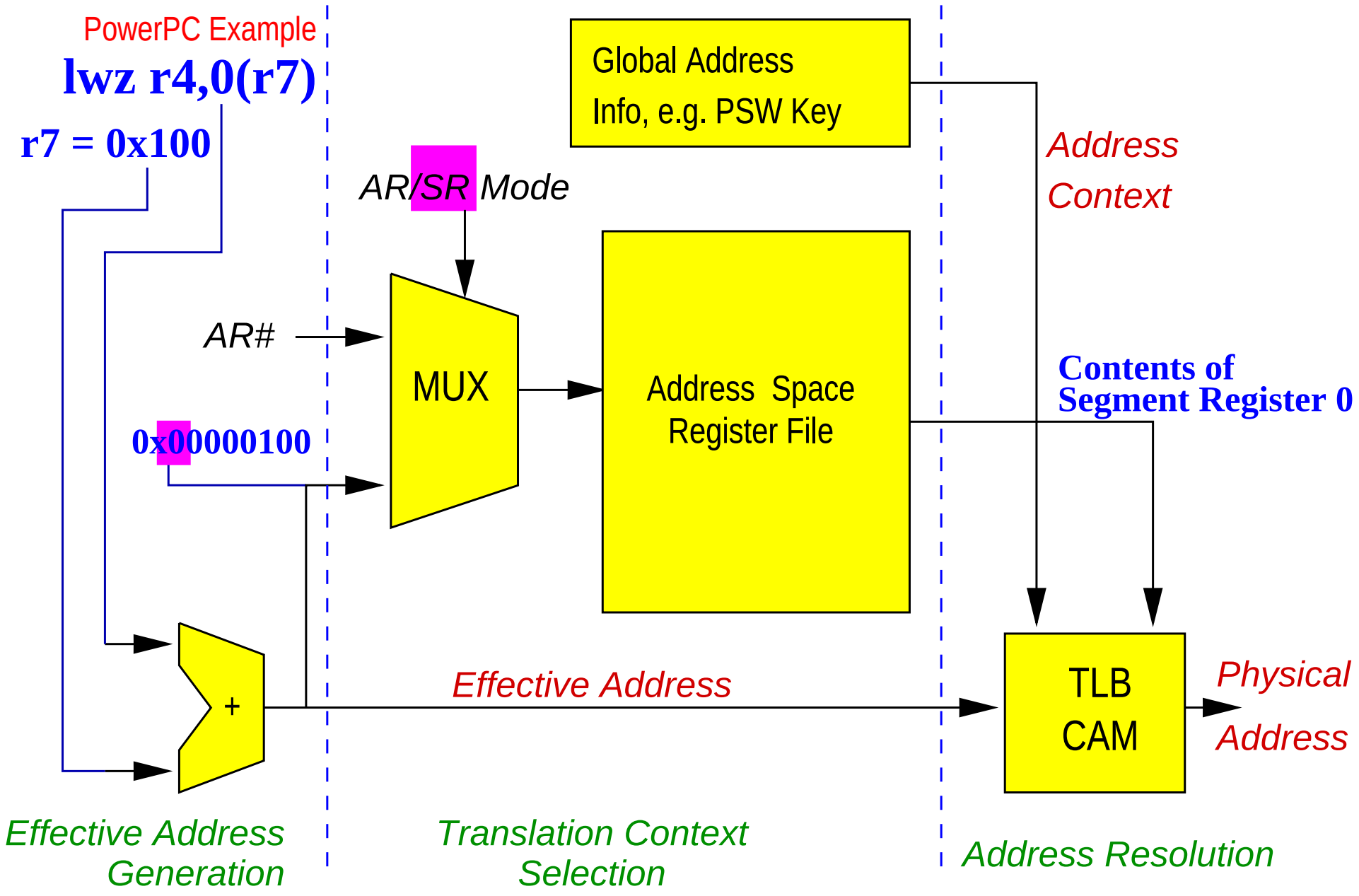
Address Translation Commonality



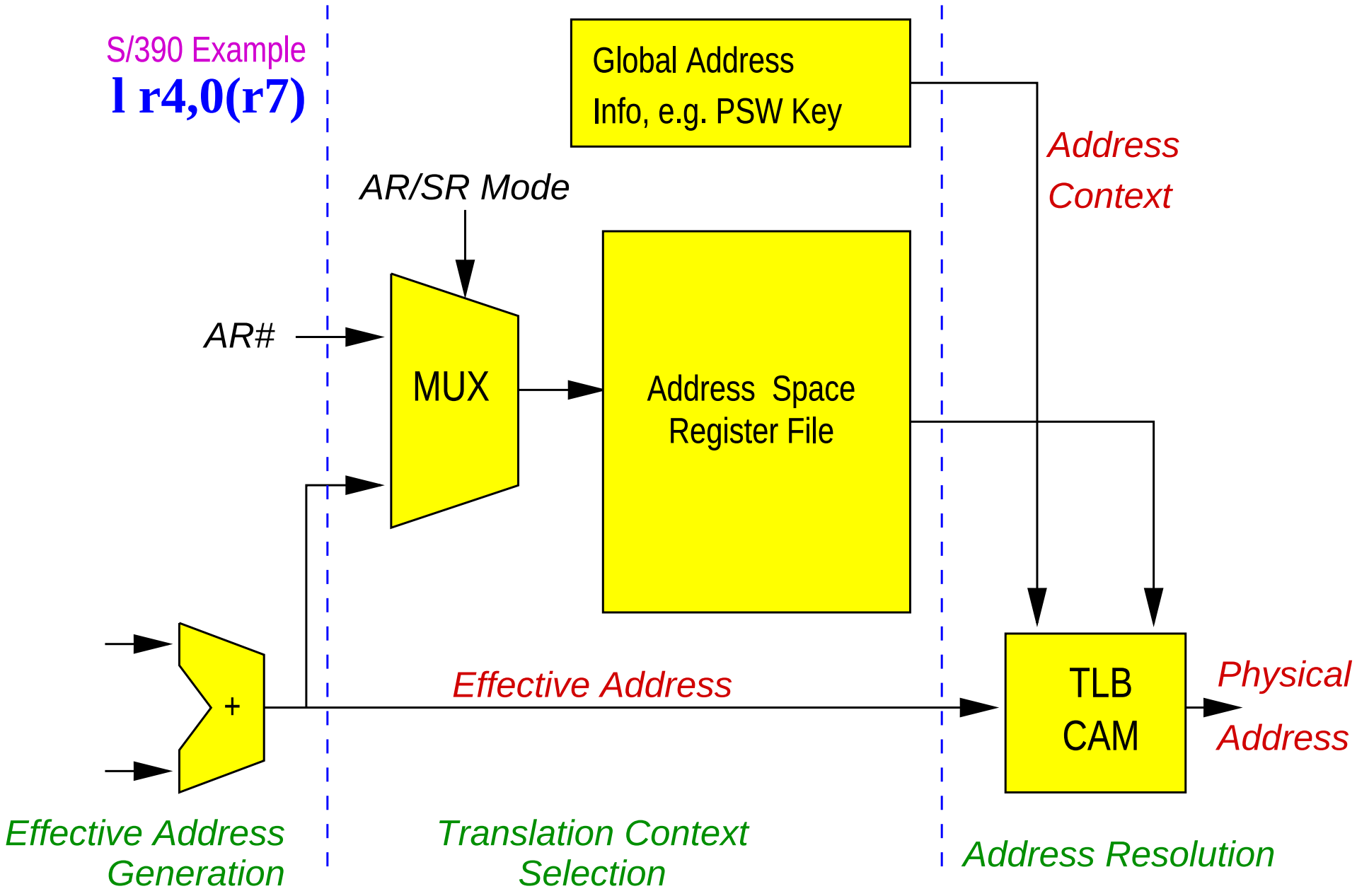
Address Translation Commonality



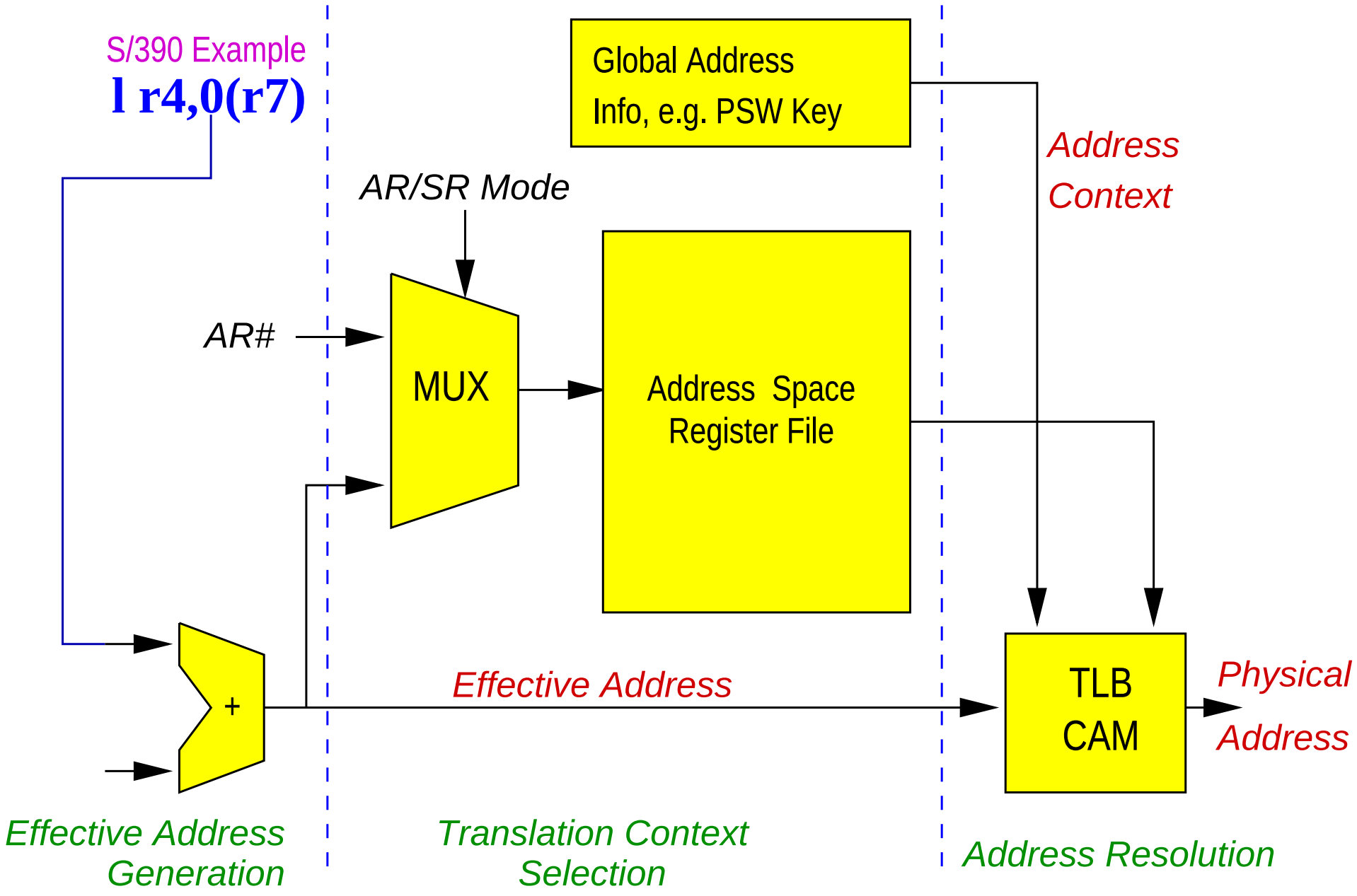
Address Translation Commonality



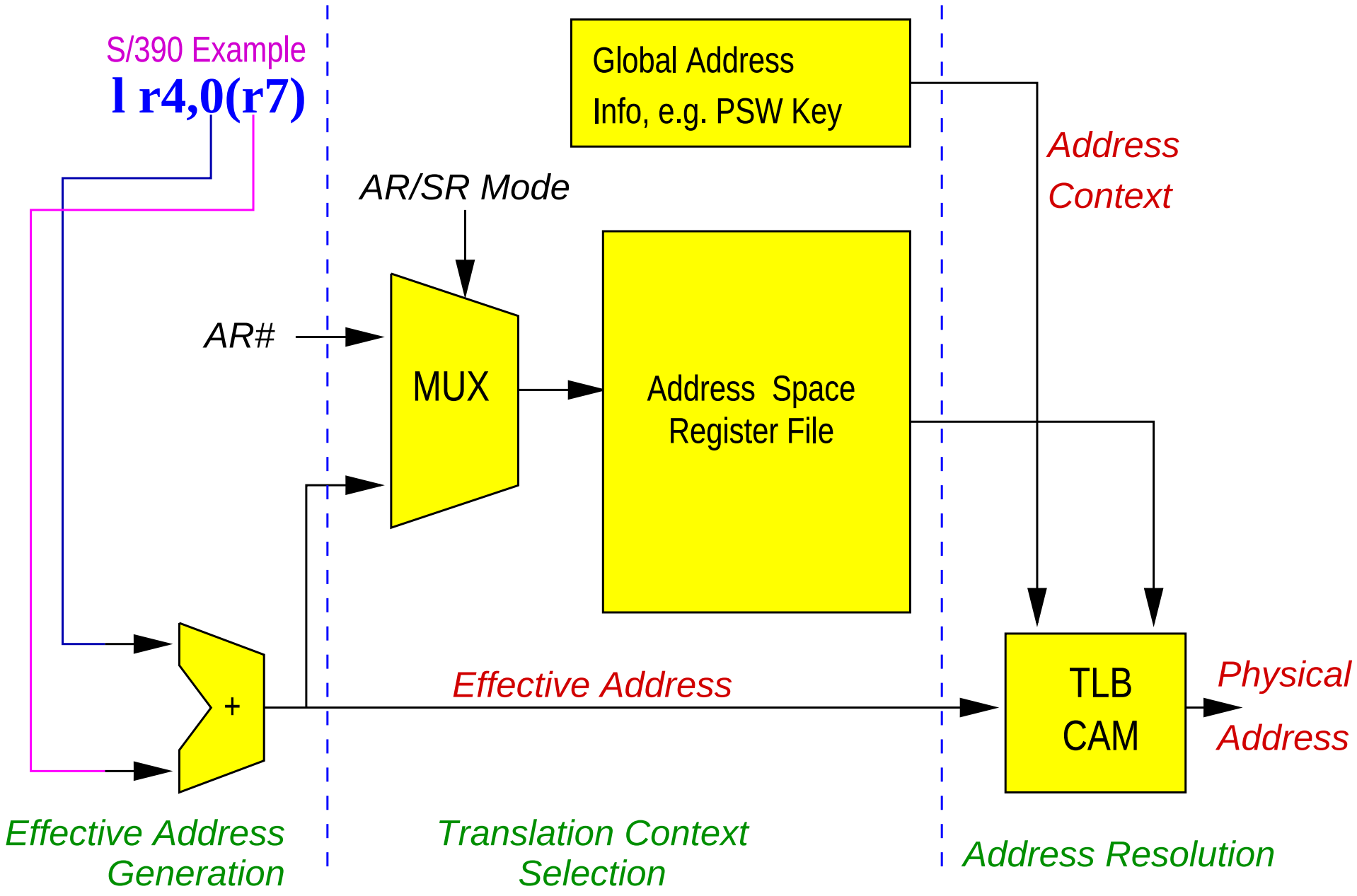
Address Translation Commonality



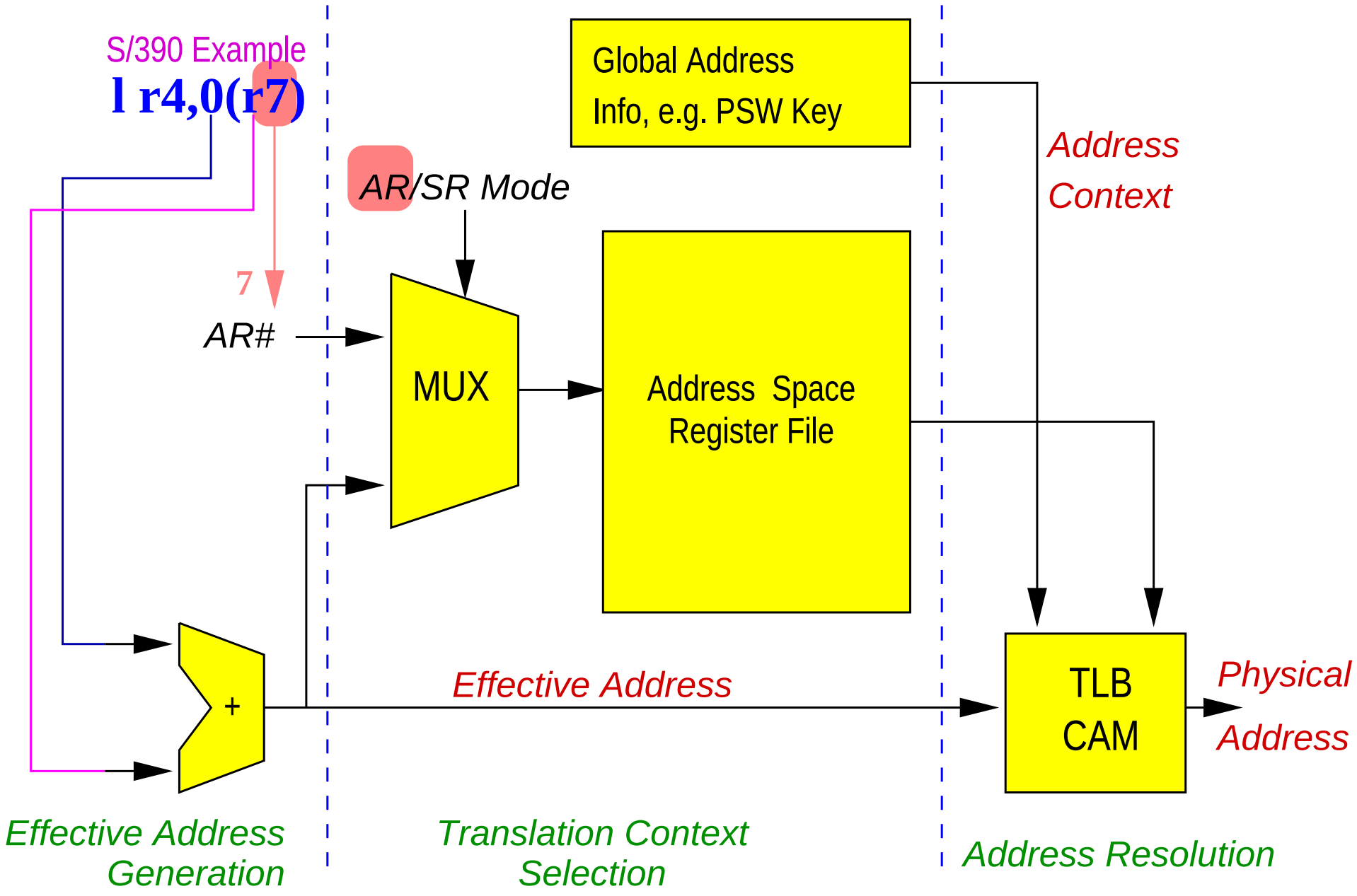
Address Translation Commonality



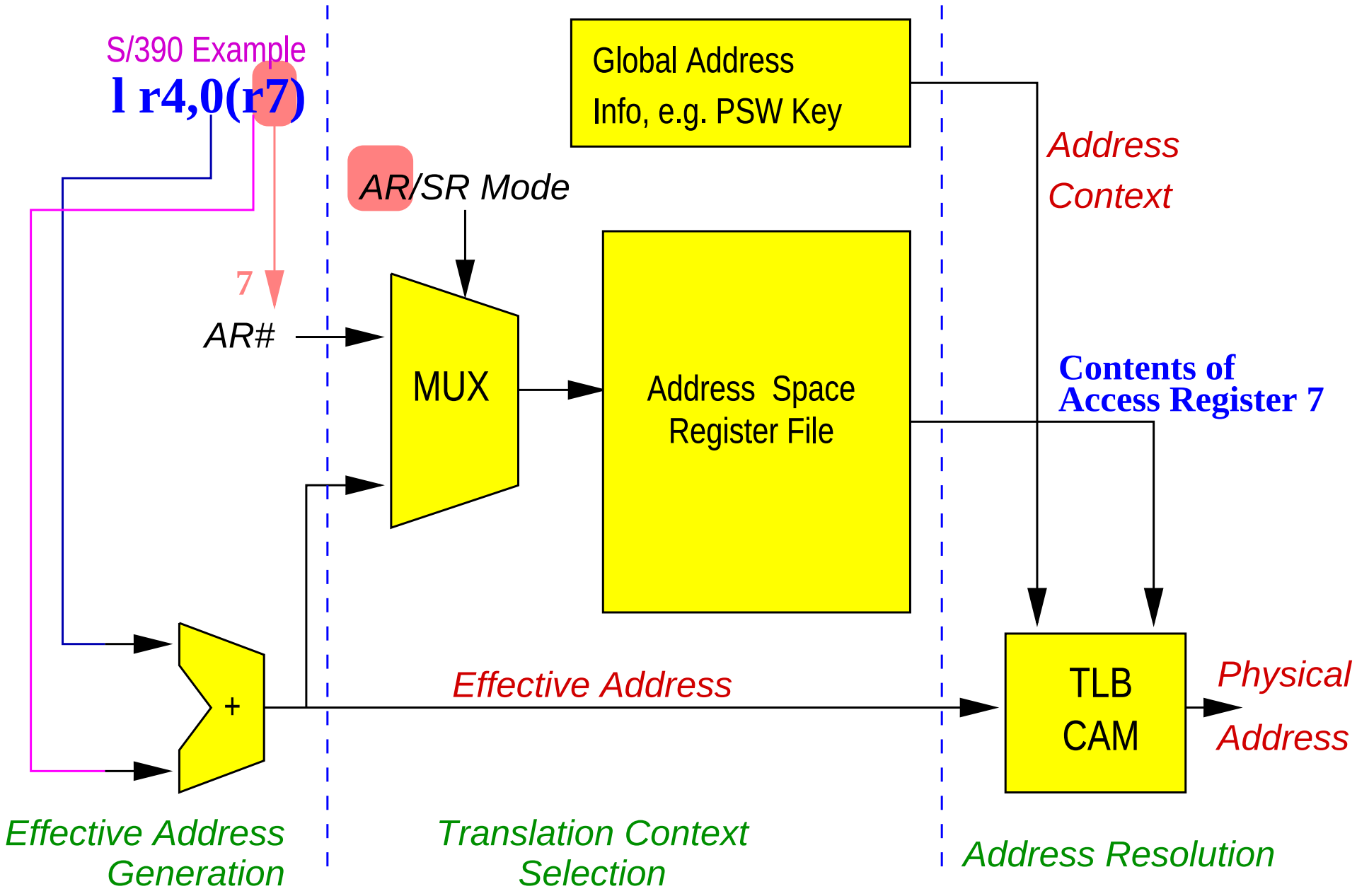
Address Translation Commonality



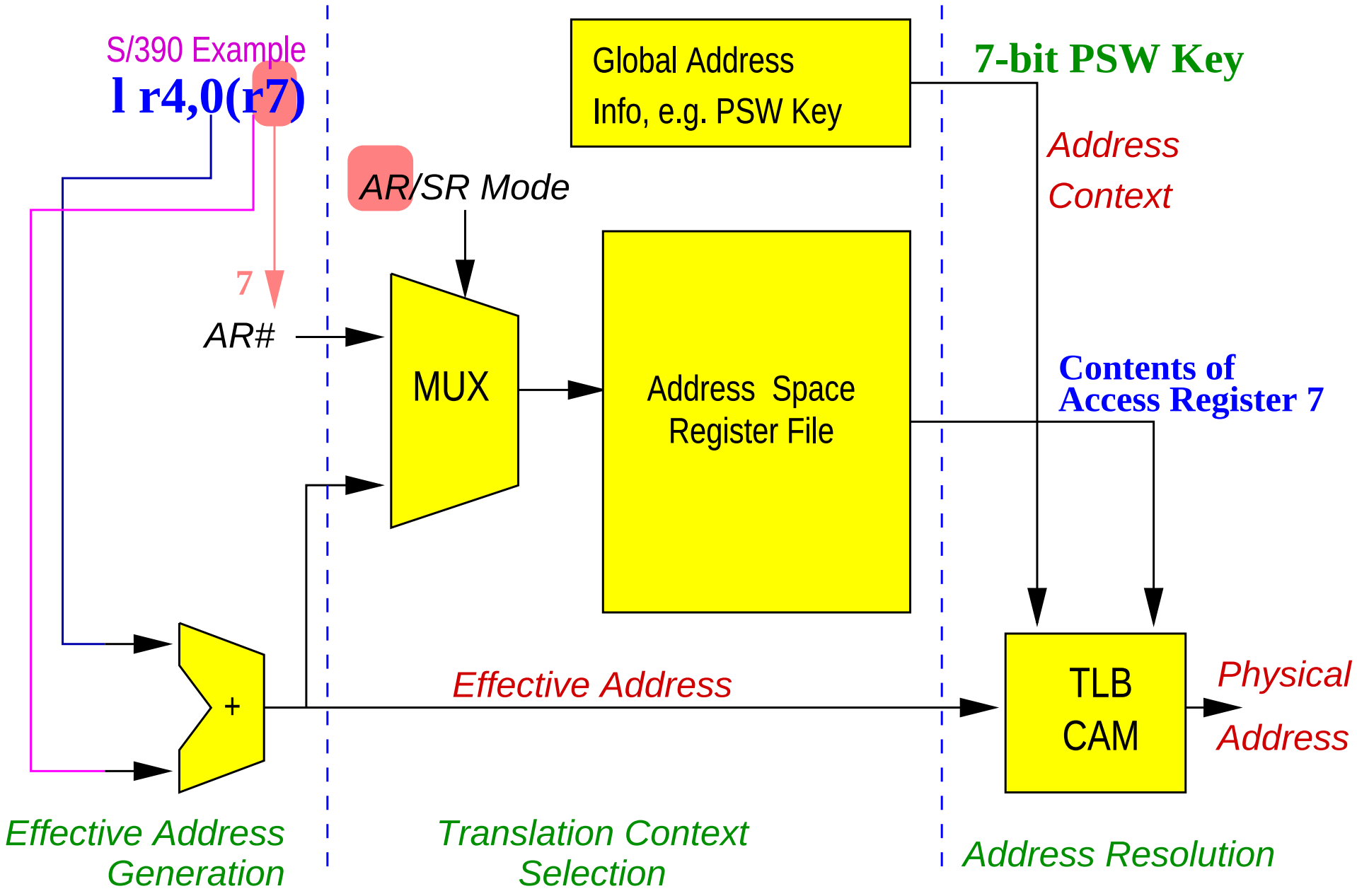
Address Translation Commonality



Address Translation Commonality



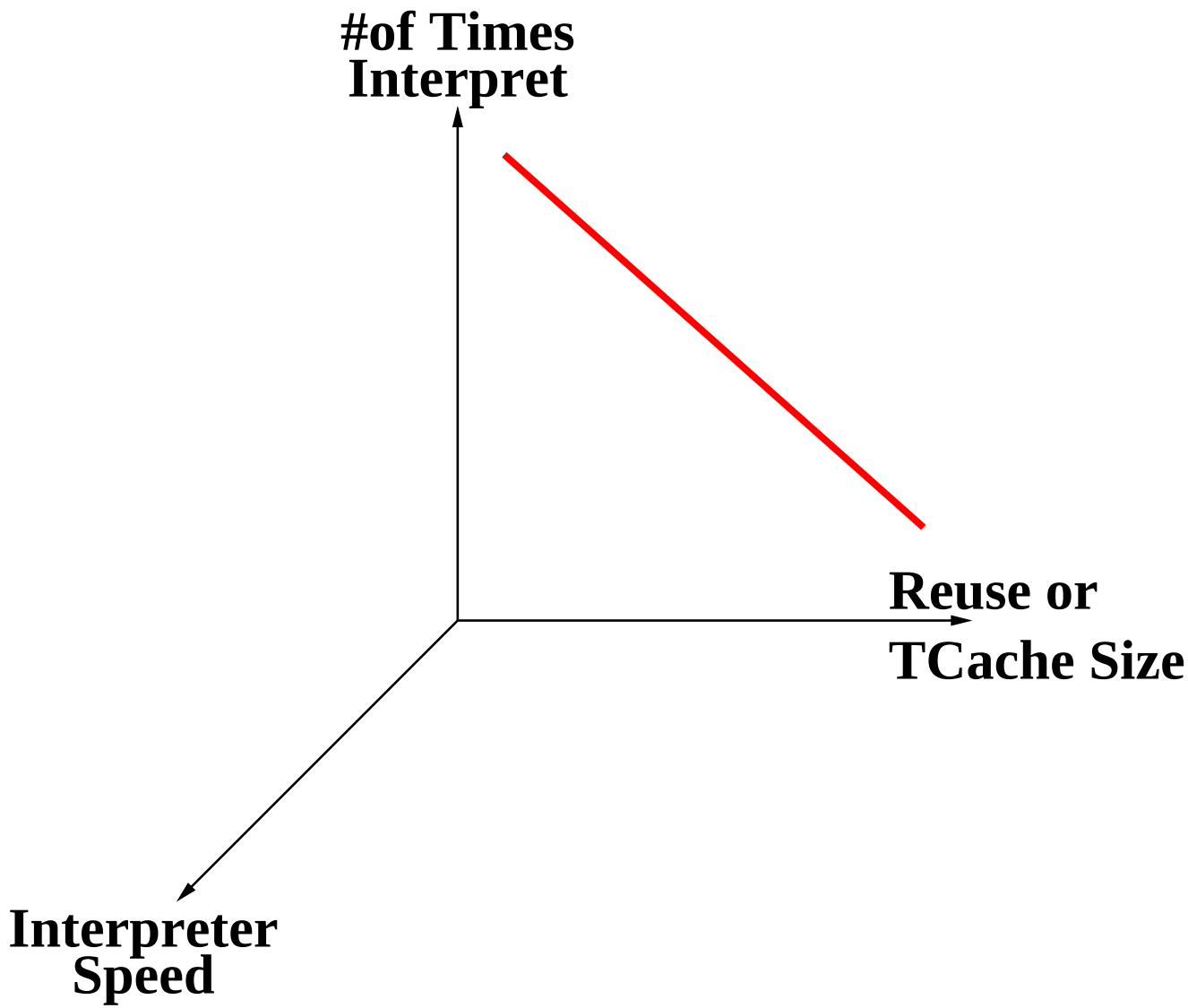
Address Translation Commonality



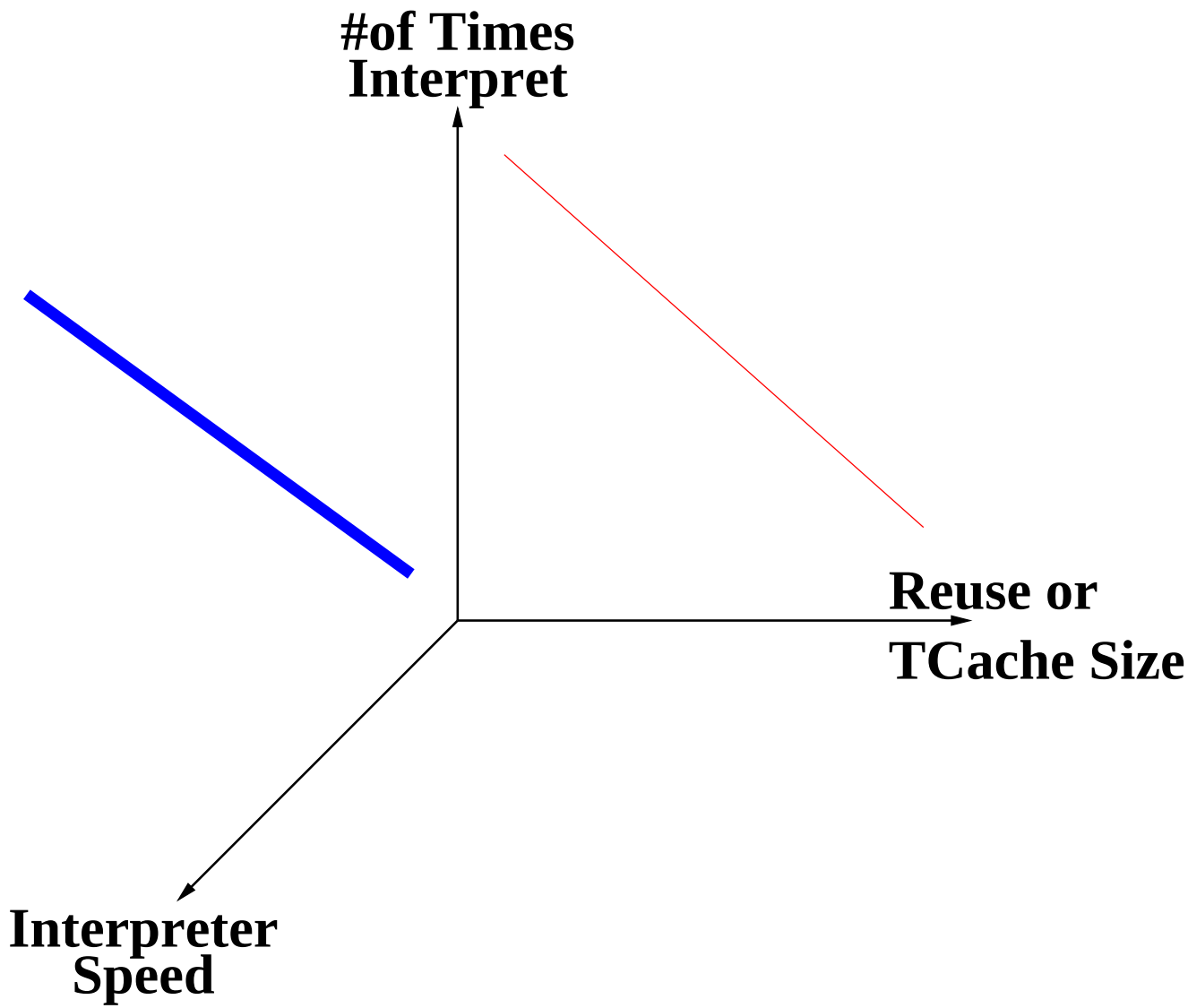
Importance of Reuse

- High code reuse $\Rightarrow$ Time consuming high optimization.
- Large **TCaches** allow more reuse at the cost of memory and lower adaptability to code changes.
- Slow translators and interpreters require high code reuse to amortize the time they take.

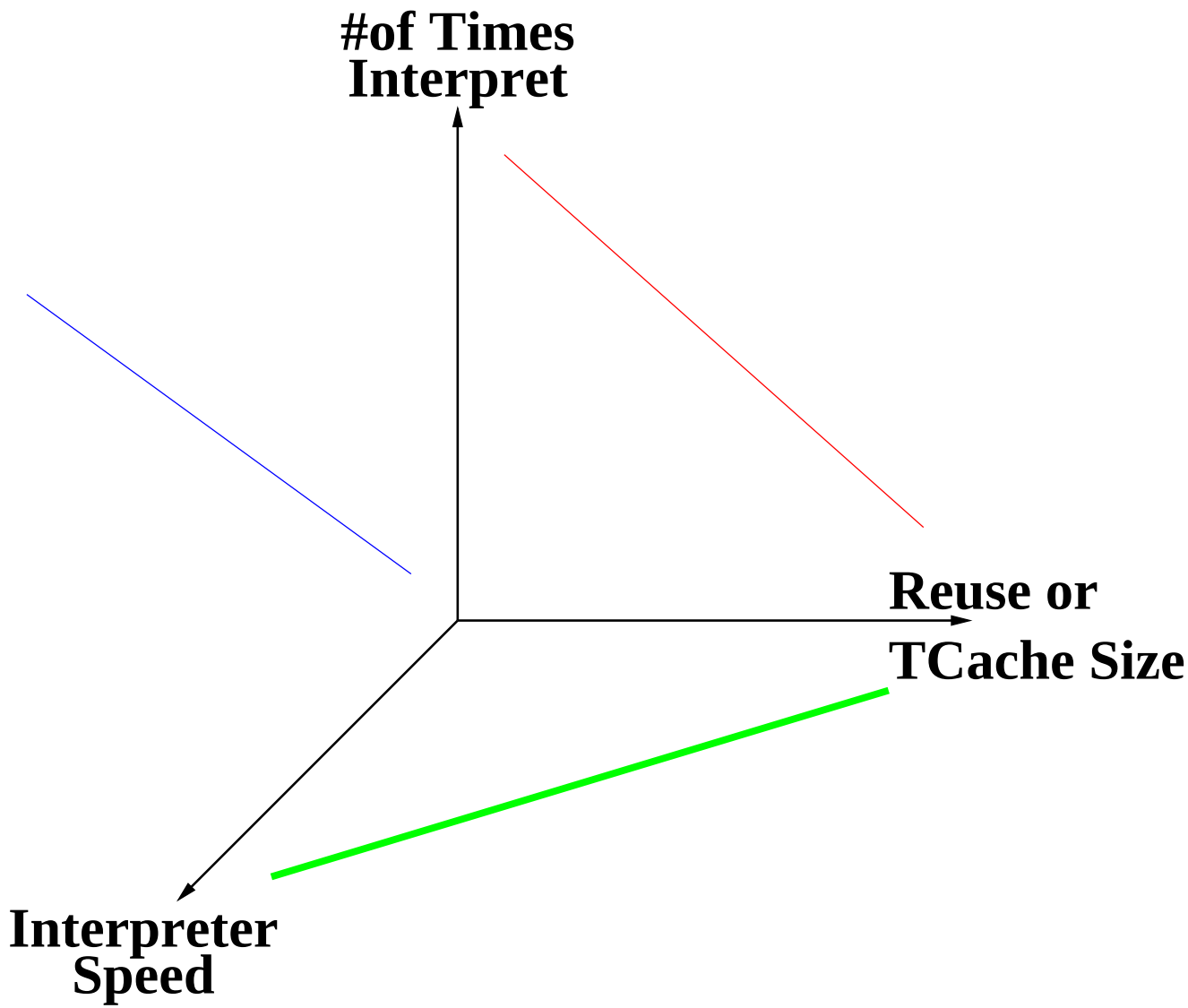
Reuse vs Interpretation & Interpretation Speed



Reuse vs Interpretation & Interpretation Speed



Reuse vs Interpretation & Interpretation Speed



Dynamic Translation Spectrum: Reuse

- **Pentium Pro** style hardware cracking in pipeline: No reuse of translation.
- **DIF** or **Pentium IV** style hardware scheduling: Reuse of a small 32K cache of translations.
- **Dynamo** software scheduling: Reuse in a 500K memory cache.
- **Crusoe** software scheduling: Reuse in a 16M memory cache.
- **DAISY** software scheduling: Reuse in a 100M memory cache.

Timetable for Micro-33 Tutorial on

Dynamic Binary Translation and Optimization

Wednesday, December 13, 2000

2:30 - 2:50	Kemal Ebcioglu:	Future Challenges
2:50 - 2:55	Erik Altman:	DAISY Demo
2:55 - 3:20	Erik Altman:	Binary Translation Issues
3:20 - 3:35	**Break**	
3:35 - 5:00	Erik Altman:	DAISY, Crusoe, Dynamo
5:00 - 5:15	Break	
5:15 - 5:45	Kemal Ebcioglu:	LaTTe