SMT – does not mean Supermarket Technology

Mario Held (mario.held@de.ibm.com)
Linux on z Systems Performance Analyst
IBM Corporation

Acknowledgements
I wish to acknowledge the help provided by Christian Ehrhardt, Linux on z Systems performance specialist

Session 17771 August 14, 2015

#SHAREorg

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

BlueMix
BigInsights
Cognos*
DB2*
DB2 Connect
Domino*
DS8000*
ECKD
FICON*
FileNet*
FlashSystem
GDPS*
GPFS
IBM*
Ibm.com
IBM (logo)*
IMS
Informix*
InfoSphere
Maximo*
MQSeries*
Performance Toolkit for VM
POWER*
Quickr*
Rational*
Sametime*
Smarter Cities*
Smarter Analytics
SPSS*
Storwize*
System Storage*
Tivoli*
WebSphere*
XIV*
z13
zEnterprise*
z/OS*
z Systems
z/VSE*
z/VM*

* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
Windows Server and the Windows logo are trademarks of the Microsoft group of countries.
ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

* Other product and service names might be trademarks of IBM or other companies.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. **All z13 numbers have been measured on pre GA hardware with pre GA software.**
Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here. All z13 numbers have been measured on pre GA hardware.
IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.
This information provides only general descriptions of the types and portions of workloads that are eligible for execution on Specialty Engines (e.g, zIIPs, zAAPs, and IFLs) ("SEs"). IBM authorizes customers to use IBM SE only to execute the processing of Eligible Workloads of specific Programs expressly authorized by IBM as specified in the “Authorized Use Table for IBM Machines” provided at www.ibm.com/systems/support/machine_warranties/machine_code/aut.html (“AUT”). No other workload processing is authorized for execution on an SE. IBM offers SE at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types and/or amounts of workloads as specified by IBM in the AUT.

Agenda

- Your CPU is a supermarket
  - Clock
  - TLB
  - Cache
  - µOps
  - Superscalar
  - Instruction grouping
  - Out of Order execution
  - Symmetric MultiProcessing
    - Shared resources
    - Locality
  - Predictions
  - Simultaneous MultiThreading
  - Specialty Engines and Spares
  - Full CEC
  - Virtualization
- SMT for Linux on z Systems
  - Once more with theory
  - SMT resources
  - Additional complexities
    - Hypervisor scheduling complexities
    - ETR / ITR – be careful with calculations
    - Additional memory footprint
    - Additional overhead
    - Utilization 50, 100, 200%?
    - Scheduling spill and fill
  - Appearance in a Linux System
  - Benchmark examples
    - The good
    - The bad
    - And the ugly

Your CPU is a supermarket

- Basics: Imagine your CPU is a supermarket
  - An instruction is represented by a cart
    - Different products are in a cart so we have to deal with different instructions
  - Carts line up in front of the till like instructions to a pipeline
  - Execution unit in our analogy is the cashier
  - Shelves are main storage
    - Positions of the products are the real addresses
    - The price tag is the data stored

- Important note
  - Many technologies are skipped intentionally
  - Things in order of explainability, not in order when they were invented
  - Intentionally a long strange trip along issues and solutions

Your CPU is a supermarket

Main Storage

Runs and searches the price

Execution

Pipeline

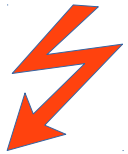
Instruction Instruction Instruction

Your CPU is a supermarket → Clock

Main Storage

Runs and searches the price

Execution



Pipeline

Instruction Instruction Instruction

Your CPU is a supermarket - Clock

- It turned out that things went wrong
  - Employees were lazy
  - Conveyor belt dropped things off till which came in too fast

  → Get fixed by introduction of global clocks ticks to synchronize everybody
  - Each Clock tick the execution unit would try to handle one instruction
  - Each Clock tick everyone would try to move one step forward

Your CPU is a supermarket - Clock

Main Storage

Clock

Runs and searches the price

Pipeline

Execution

Instruction Instruction Instruction

Your CPU is a supermarket → Cache

Main Storage

Clock

Runs and searches the price

Pipeline

Execution

Instruction Instruction Instruction

Your CPU is a supermarket - Cache

- There is no computer based till in your market and the cashier can't remember prices
  - You need a Cache represented by a chalkboard
    - Holds prices for the products already fetched
    - If product found in the cache the executions runs much faster
  - Cache is indexed by its position in the market (=> real addresses)

Your CPU is a supermarket - Cache

Runs and searches
the price **if not in cache**

Execution

Instructions

Your CPU is a supermarket → TLB / Load-Store Unit

Runs and searches
the price **if not in cache**

Execution

Instructions

Your CPU is a supermarket – Transaction Lookaside Buffer TLB / Load Store Unit LSU

- Load / store unit writes to the Chalkboard Cache
  - Fetch prices by running into the market and checking the label
  - Bring new labels to the shelf if prices change due to execution
  - On the next charts represented by a running dog
- The dogs are not super-smart, so they don't know where to look
  - Information were to search is required

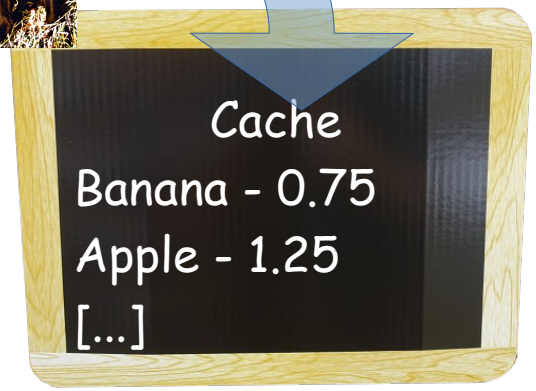
Your CPU is a supermarket - LSU

Load / Store Unit

Fetch

Write

? ?
? ?



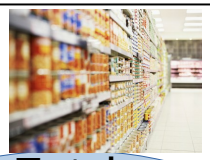
Execution

Instructions

Your CPU is a supermarket – TLB / LSU

- Customers refer to the same products with different names (apple, “Golden Delicious”, “Fuji”, ...)
  - Different names are here analogies for different virtual addresses pointing to the same physical address
    - Virtual addresses can get resolved by walking indexes in a huge book
    - In a CPU these indexes are the page tables
  - Another Chalkboard hold places where to find products (TLB)
    - Done ahead of actual execution as e.g. LSU needs it
  - TLB is indexed by the names the customer use (=> virtual addresses)
  - Remember: Cache is indexed by its position in the market (=> real addresses)

Your CPU is a supermarket - TLB



Fetch

Load/Store Unit

Read

Write

Address Translation

Write

TLB

Apple - Shelf A Column 12

Banana – Shelf C Column 7

Pink lady – Shelf A Column 12

[…]

Cache

Shelf A Column 12 - 0.75

Shelf C Column 7 - 1.25

[…]

Watch

Execution

Remember – this is a pipeline:

Early: Address resolution Later: Data Fetching Eventually: Execution

Instructions

Your CPU is a supermarket – CISC and µOps

- The conveyor belt was soon too full
  - But often people bought the same group of things like chips and beer
  - That is true for some instructions as well, usually being used together
  - The market created products-sets in special boxes
  - Like complex instructions (CISC)
  - Some other markets pretended to sell product-sets but actually give you only a better price if you buy products together like it handles instructions together keep the execution units more trivial (µOps)
  - Microcode is way more complex than µOPS, more like a full shopping list

Your CPU is a supermarket → CISC and µOps

Load/Store Unit

Fetch

Read

Write

Address Translation

Write

TLB

Apple - Shelf A Column 12

Banana – Shelf C Column 7

[...]

Cache

Shelf A Column 12 - 0.75

Shelf C Column 7 - 1.25

[...]

Watch

Instructions

Execution

Your CPU is a supermarket → CISC and µOps

Load/Store Unit

Fetch

Read

Write

Address Translation

Write

TLB

Apple - Shelf A Column 12

Banana – Shelf C Column 7

[...]

Cache

Shelf A Column 12 - 0.75

Shelf C Column 7 - 1.25

[...]

Watch

Instructions

Execution

New Complex instruction doing all in one (CISC)

Your CPU is a supermarket → CISC and µOps

Load/Store Unit

Fetch

Read

Write

Address Translation

Write

TLB

Apple - Shelf A Column 12

Banana – Shelf C Column 7

[...]

Cache

Shelf A Column 12 - 0.75

Shelf C Column 7 - 1.25

[...]

Watch

Instructions

Execution

broken into µOps to simplify execution

Your CPU is a supermarket → Superscalar

Load/Store Unit

Fetch

Read

Write

Address Translation

Write

TLB

Apple - Shelf A Column 12

Banana – Shelf C Column 7

[...]

Cache

Shelf A Column 12 - 0.75

Shelf C Column 7 - 1.25

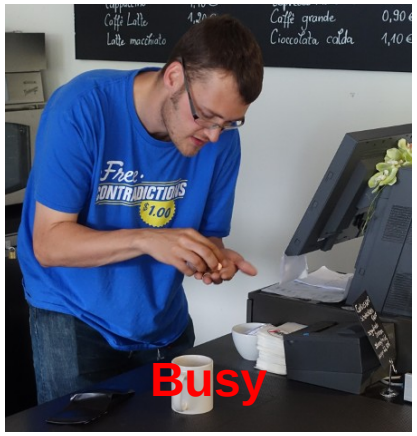
[...]

Watch

Instructions

Complex Instruction

Execution slows things down



Your CPU is a supermarket - Superscalar

- A single cashier isn't enough
  - One needs to handle the products one by one
  - BUT – there is not enough space to fit multiple cashiers in person
- Specialized Units to process products (→ multiple execution units)
  - Weighing scales for the fruits
  - Automated Scanners for barcodes
  - Self service stations

    → These can work on products concurrently

Your CPU is a supermarket - Superscalar

Multiple Execution Units (specialized)

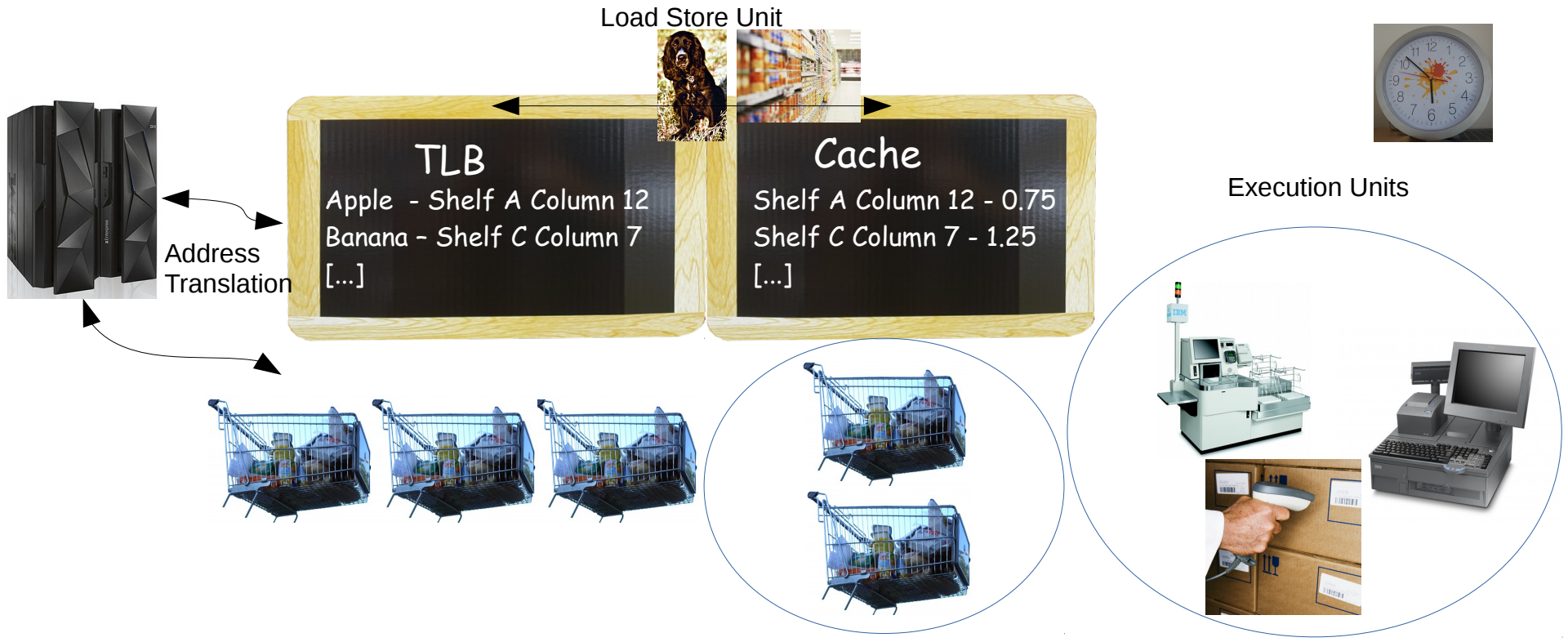
Your CPU is a supermarket → Instruction grouping

Your CPU is a supermarket – Instruction grouping

- Of the Multiple execution units some usually are idle
  - Change the process, allow to handle multiple carts at once
  - One has to check that the customers might don't surpass each other
    - But not interfere what they buy

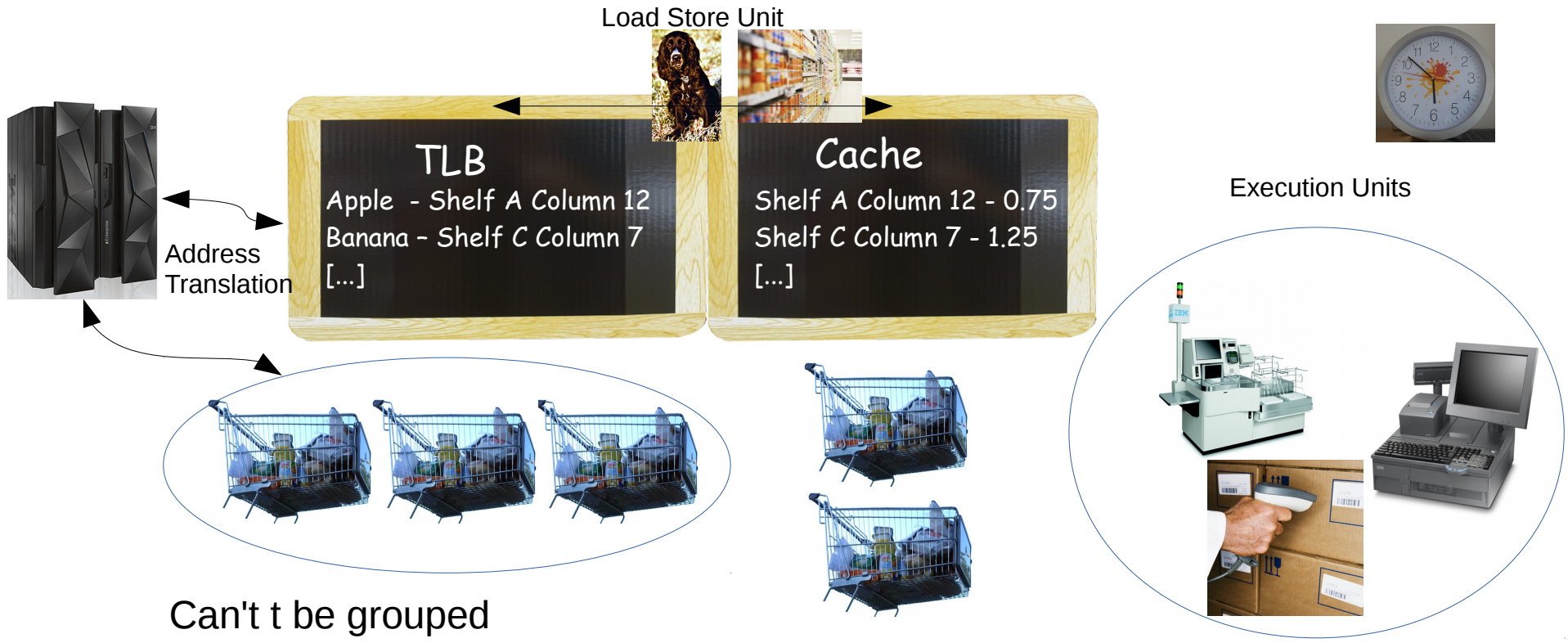
  → this is instruction grouping

Your CPU is a supermarket – Instruction grouping



Two or more can be handled at once now

Your CPU is a supermarket → Out of Order

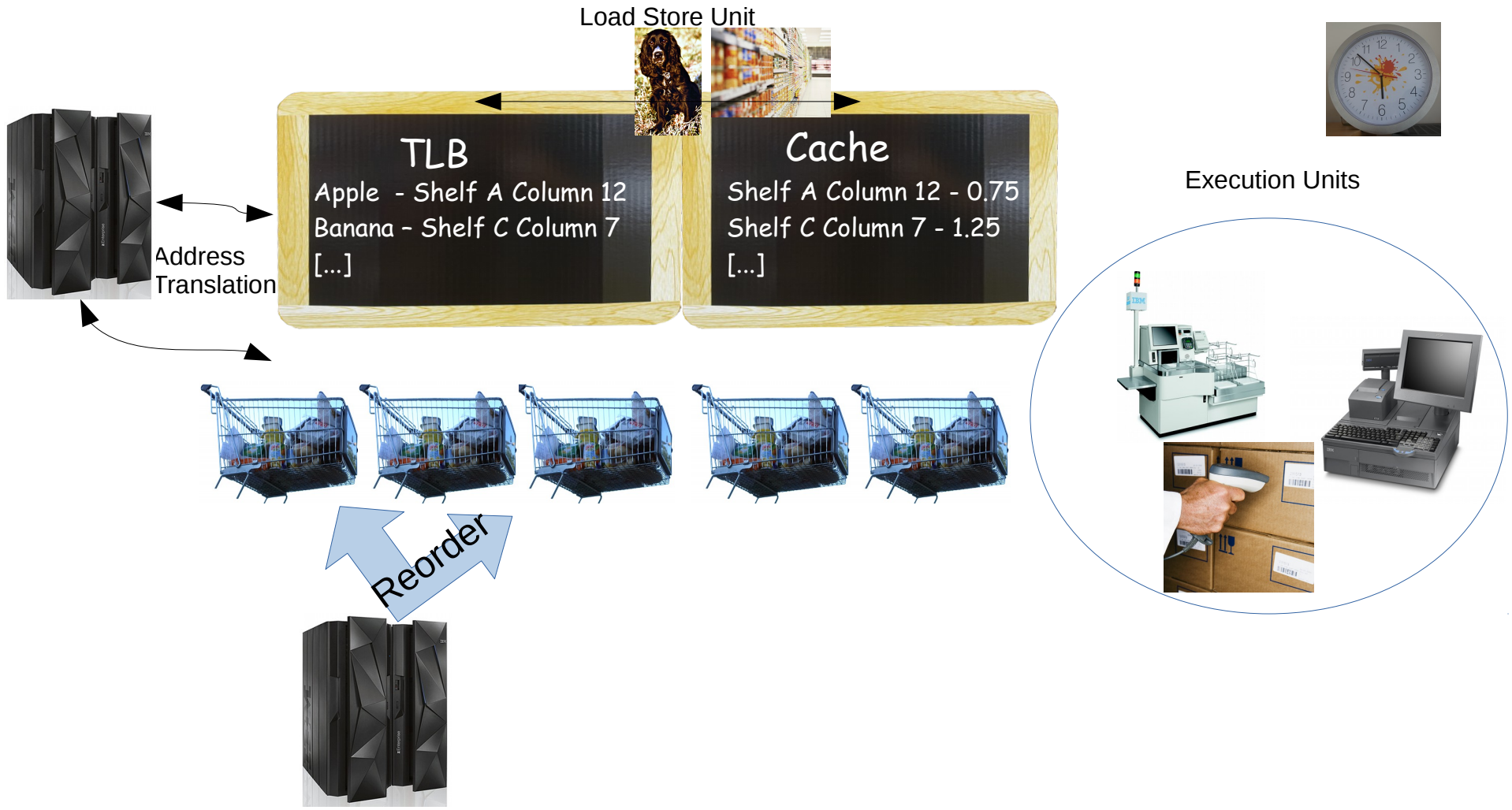


Can't t be grouped
Or a later one would be in cache already but has to wait in line

Your CPU is a supermarket – Out of order execution

- Sometimes the multiple execution units and grouping could work better
- Sometimes the stuff at the front is not in Cache / TLB and has to wait
  - other products further behind in the line could be processed
  - Someone could reorder the customers to make better use of resources
    (In a real CPU that is a hard task with a lot of extra conditions to check)

Your CPU is a supermarket → Out of Order



Your CPU is a supermarket → SMP

Your CPU is a supermarket – SMP

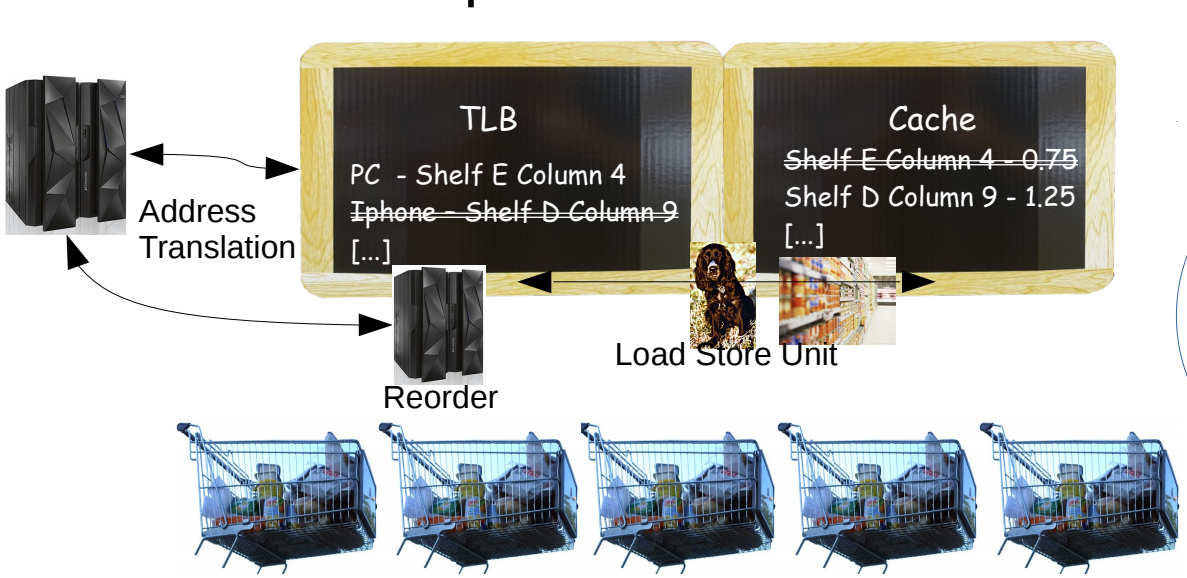
- So far we made the supermarket faster, now lets grow
  - We could also handle more customers by adding another cash register
  - This doesn't lead to two times the “throughput”
    - Via shared resources cash registers can block each other, for example if one calls the electronics department to clarify something the line is busy for the other

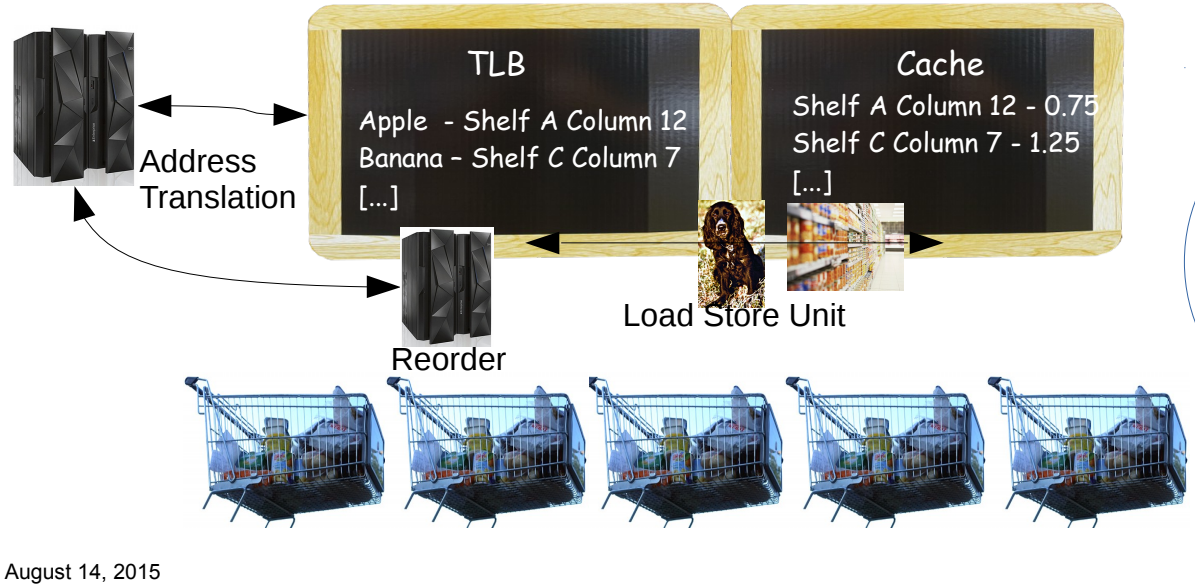
      → Locks for single resources (Logging ...)
    - Also customers now sometimes switch tills, but the chalkboard doesn't have the products ready they buy

      → Cache / TLB hotness (falling out of the sweet spot costs performance)
    - The same is true if a customer returns a product but put it to a different shelf (store)

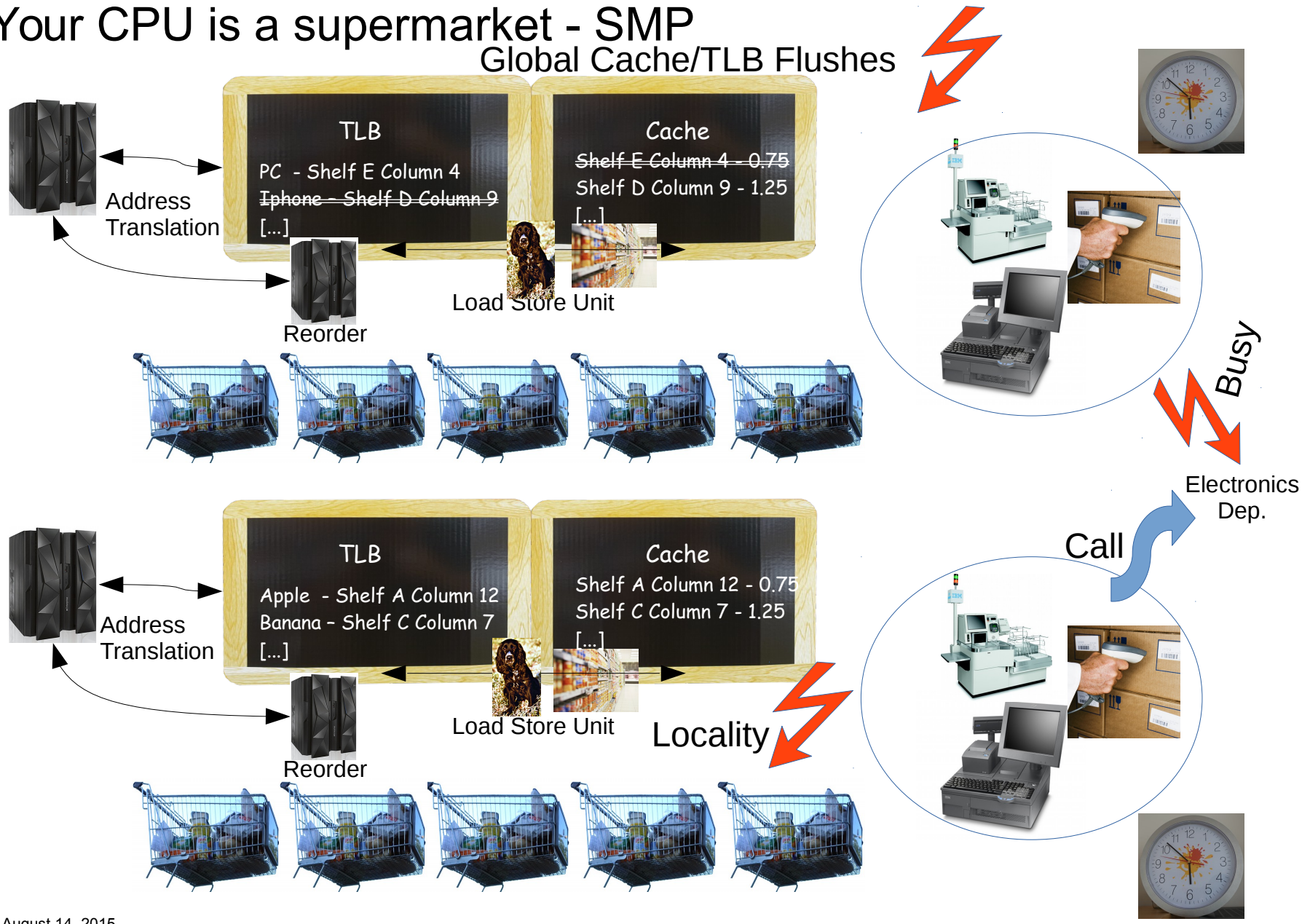
      → Cache / TLB flush and coherency in general

Your CPU is a supermarket - SMP

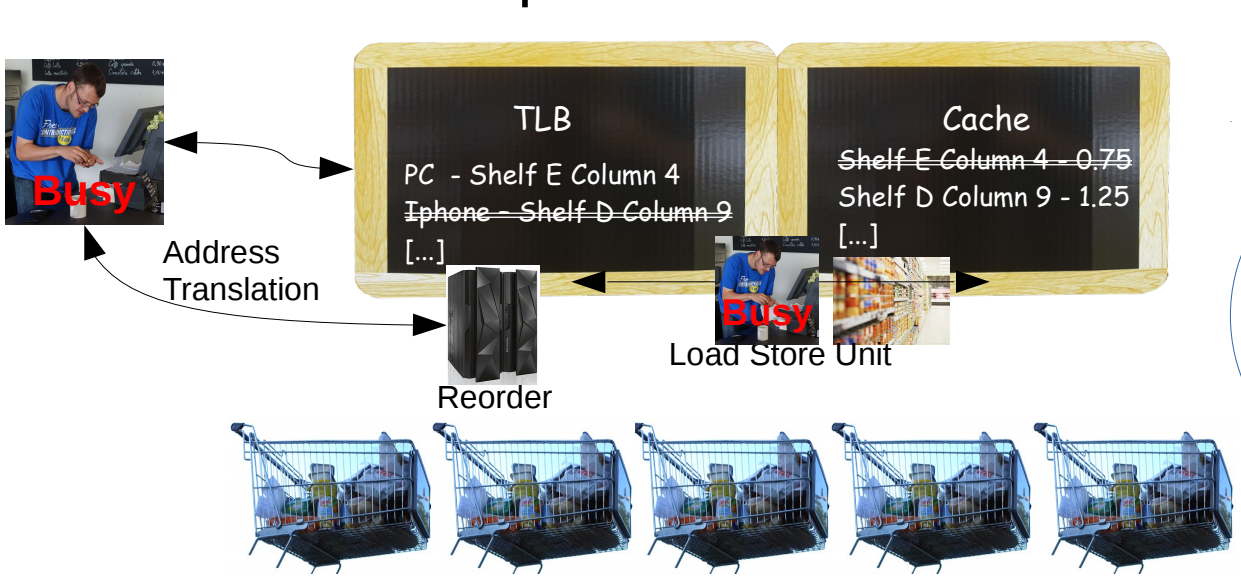


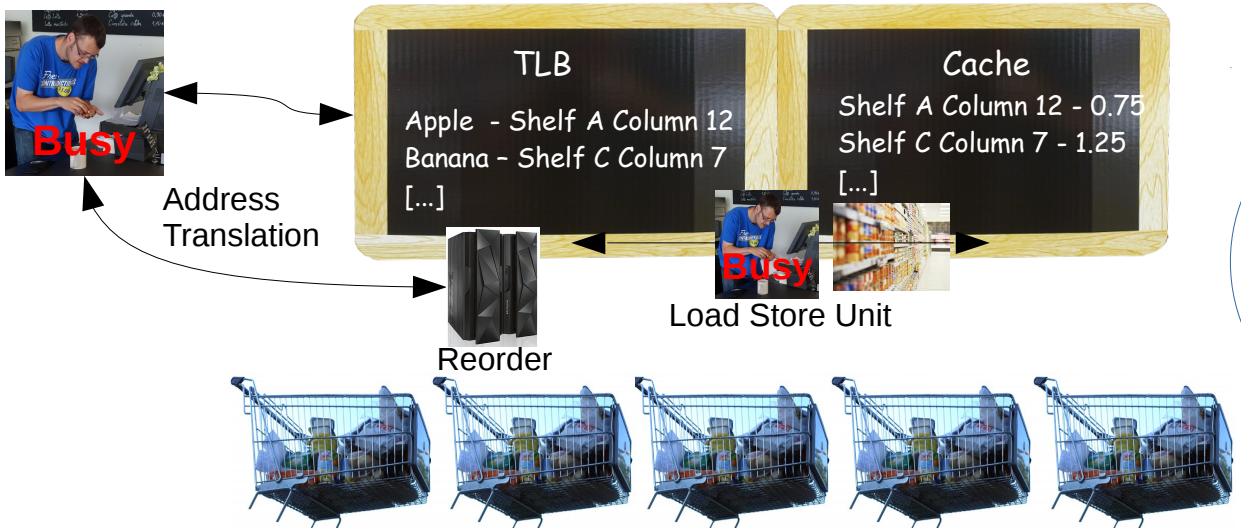


Your CPU is a supermarket - SMP



Your CPU is a supermarket → Predictions

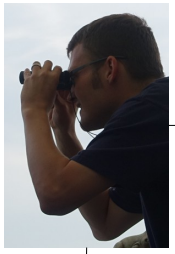




Summary - Predictions

- Reorder / TLB / ... could work better if they knew ahead what comes next
  - But some purchase decisions are made on the fly
  - So predictions where made what people will buy next
    - Pre-calculate according positions
    - Pre-fetch according prices
  - This can help TLB / Caching to keep track with ever faster execution units
  - That more or less matches Branch Predictions and all similar technologies

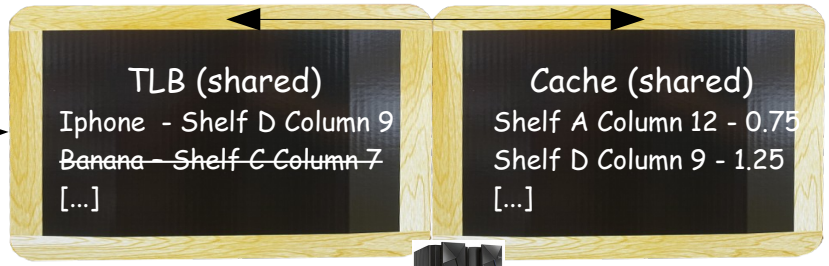
Your CPU is a supermarket – Predictions



Predictions &
Prefetch

Load Store Unit

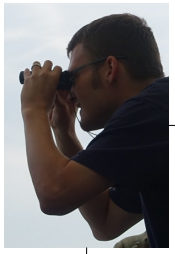
Address
Translation



Reorder

Branch predictions
Prefetch
Precalculations – before branch target is known

Your CPU is a supermarket → SMT

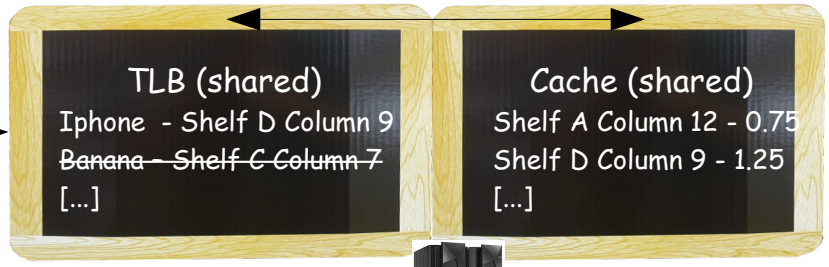


Predictions &
Prefetch

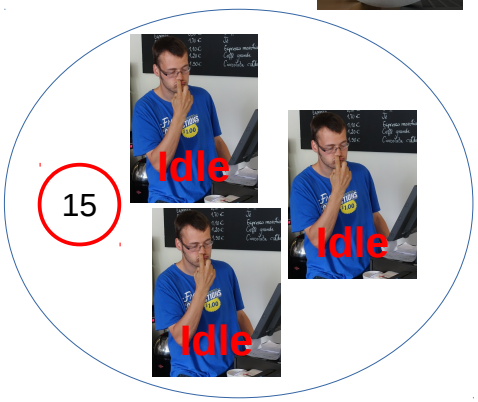
Not enough space
for more

Load Store Unit

Address
Translation



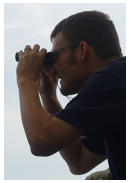
Reorder



Your CPU is a supermarket – SMT

- More tills with all equipment cost too much and don't fit (die space)
  - The supermarket wants to grow further, but there is no space
  - There could be enough space for two lines in front of the cashier (→ SMT)
    (One pipeline with a few extras handles two instruction streams)
  - Although this has far less power than another “full till”
    - Customers in both lines share the TLB / Cache Chalkboards (Trashing)
    - In a real CPU they also share more (renaming registers, decoder, ...)
    - One might wait for a relative in the other line and buys extra stuff while being bored (spinning on one thread while the other needs to free a lock)
  - Benefits vary
    → Your market can handle more customers at once
    → Even if overall throughput increases individuals will need more time to buy things
    → There is more variety now, makes is more difficult to predict how long it will take

Your CPU is a supermarket - SMT



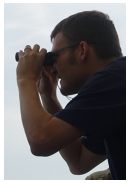
Predictions &
Prefetch

Load Store Unit

Address
Translation

Reorder

Your CPU is a supermarket - SMT



Predictions &
Prefetch

Load Store Unit

Address
Translation

TLB (shared)
Iphone - Shelf D Column 9
~~Banana - Shelf C Column 7~~
[...]

Cache (shared)
Shelf A Column 12 - 0.75
Shelf D Column 9 - 1.25
[...]

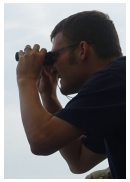
Wait

Reorder

Wait

Wait
(cross lock)

Your CPU is a supermarket → Specialty Engines



Predictions &
Prefetch

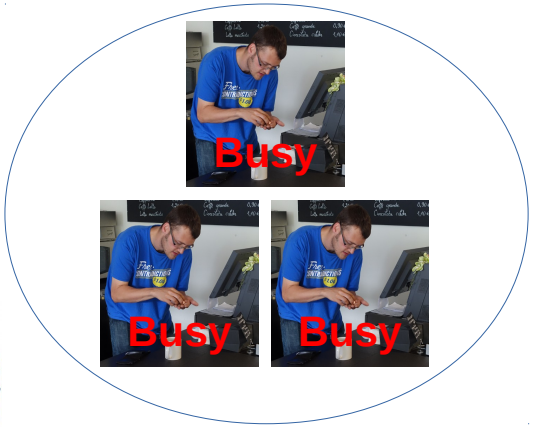
Load Store Unit

Address
Translation

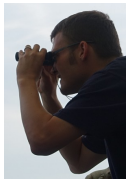
TLB (shared)
Iphone - Shelf D Column 9
~~Banana - Shelf C Column 7~~
[...]

Cache (shared)
Shelf A Column 12 - 0.75
Shelf D Column 9 - 1.25
[...]

Reorder



Your CPU is a supermarket → Specialty Engines



Predictions &
Prefetch

Load Store Unit

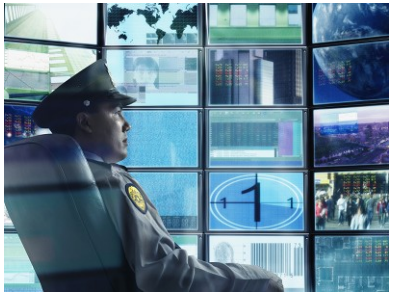
Address
Translation

TLB (shared)
Iphone - Shelf D Column 9
~~Banana - Shelf C Column 7~~
[...]

Cache (shared)
Shelf A Column 12 - 0.75
Shelf D Column 9 - 1.25
[...]

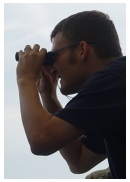
Reorder

Cryptographie – CPACF
Compression



CPACF

Your CPU is a supermarket → Full CEC



Predictions &
Prefetch

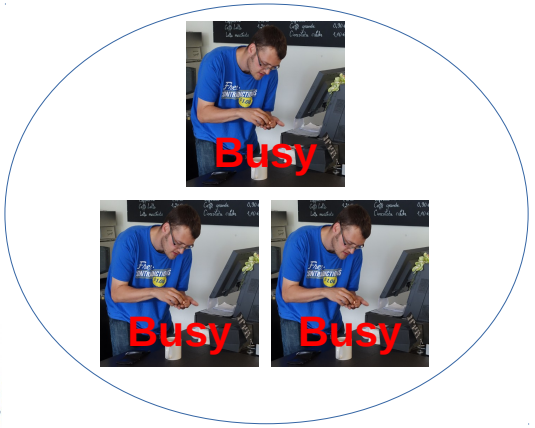
Load Store Unit

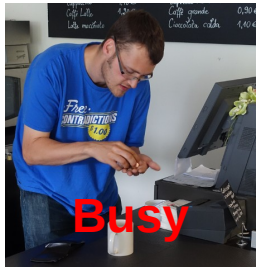
Address
Translation

TLB (shared)
Iphone - Shelf D Column 9
~~Banana - Shelf C Column 7~~
[...]

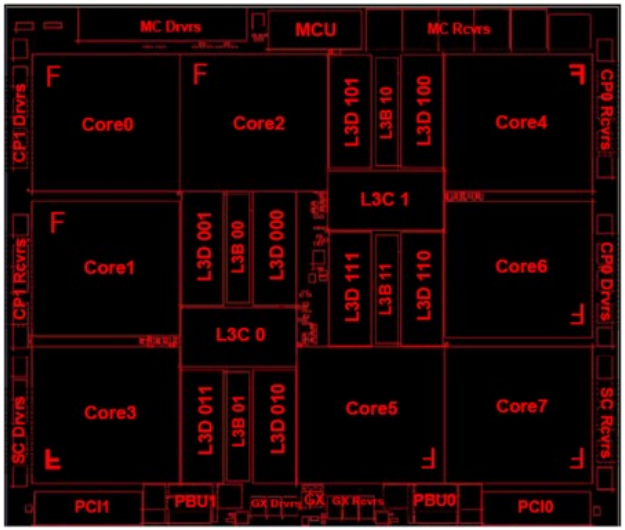
Cache (shared)
Shelf A Column 12 - 0.75
Shelf D Column 9 - 1.25
[...]

Reorder

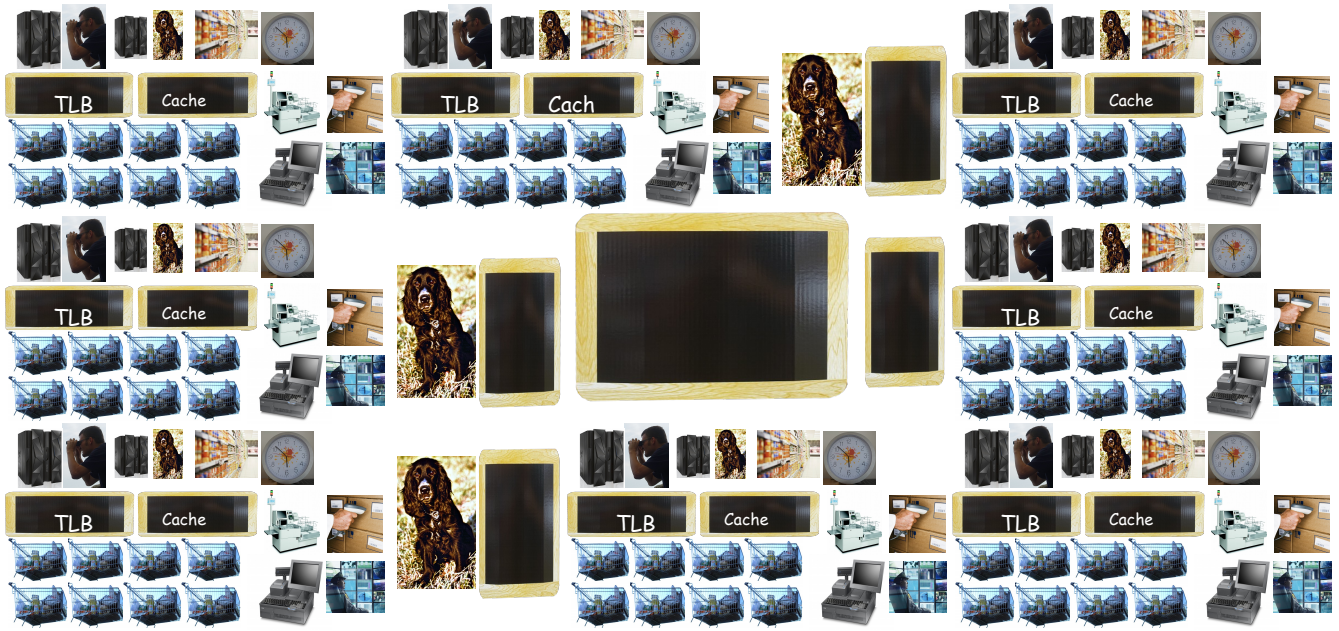




Your 8 core 16 thread chip is a modern supermarket



IBM z13 Technical Guide
http://www.redbooks.ibm.com/abstracts/sg248251.htm



Your 24 core 48 thread node is a modern supermarket

Shared Cache

Not directly End User accessible SAP / Spare / ...

Not all are good out of Manufacturing

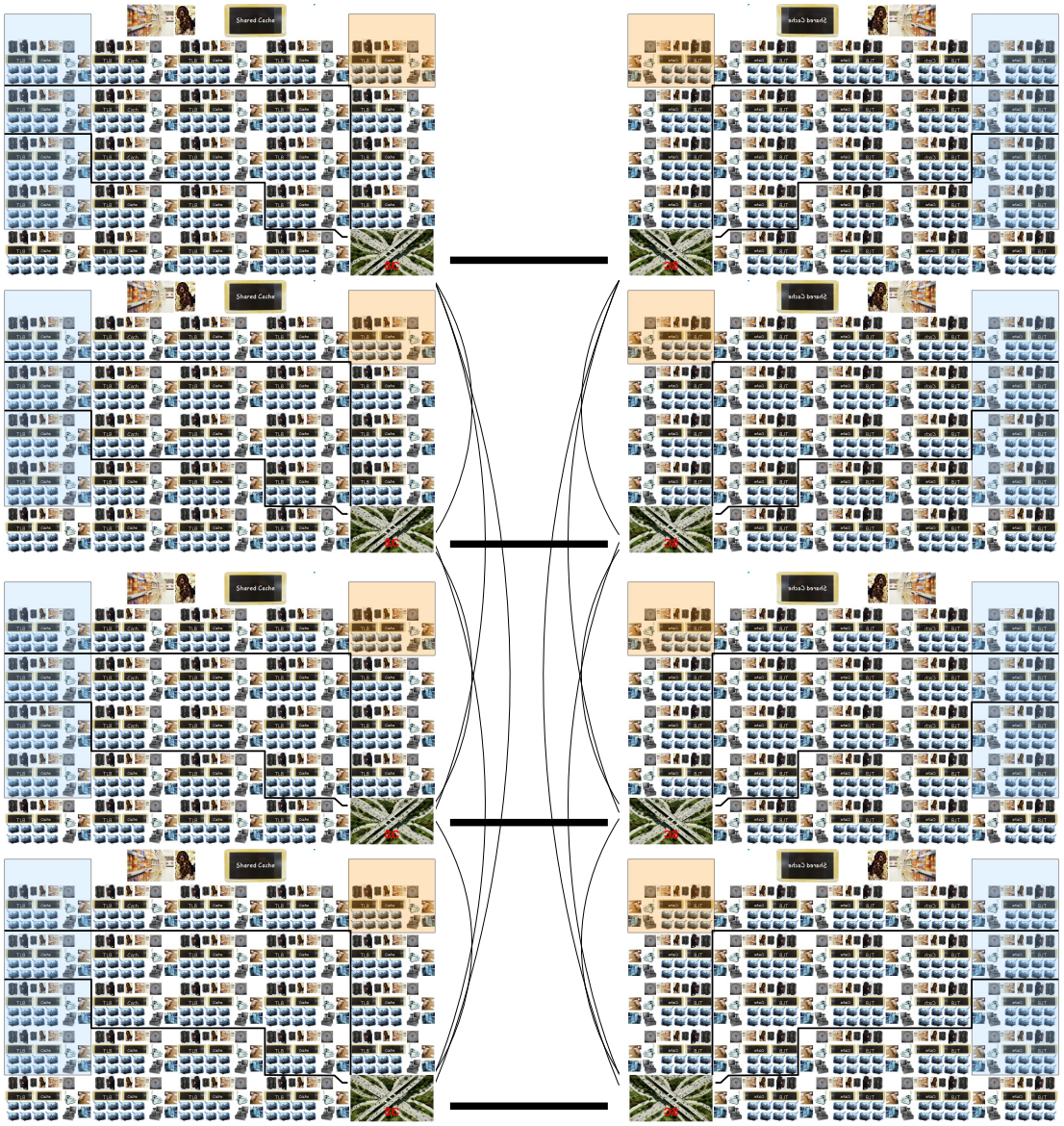
Chip

Chip

Chip

SC

Interconnectivity between nodes on IBM z13



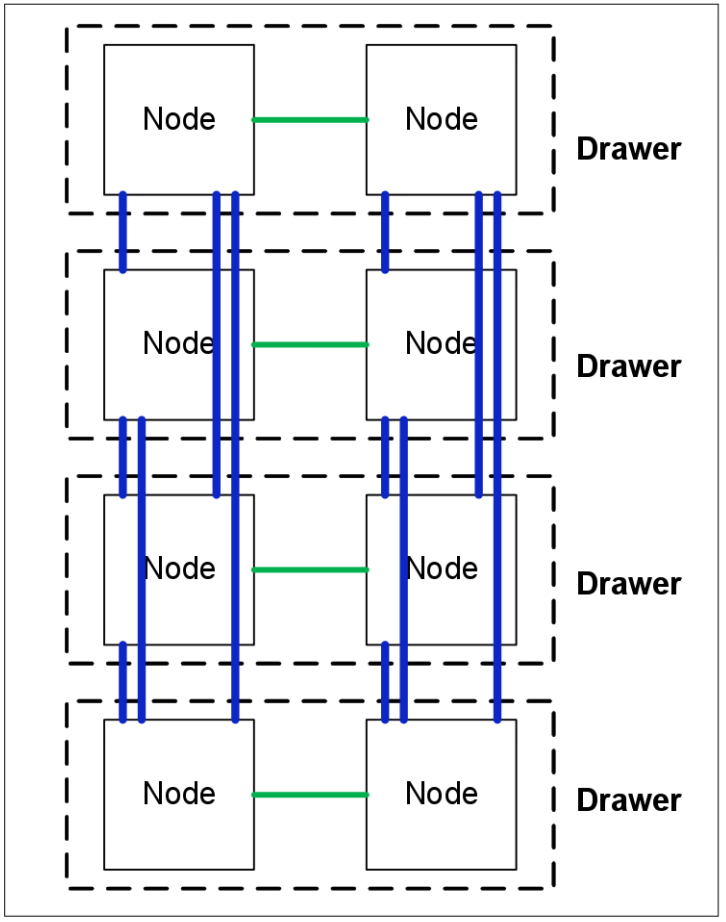


Figure 2-8 Drawer to drawer communication

IBM z13 Technical Guide
http://www.redbooks.ibm.com/abstracts/sg248251.html

All that did not yet even touch virtualization

CPU Virtualization

Different shops share the checkout area

Memory Overcommit

Shared remote warehouse

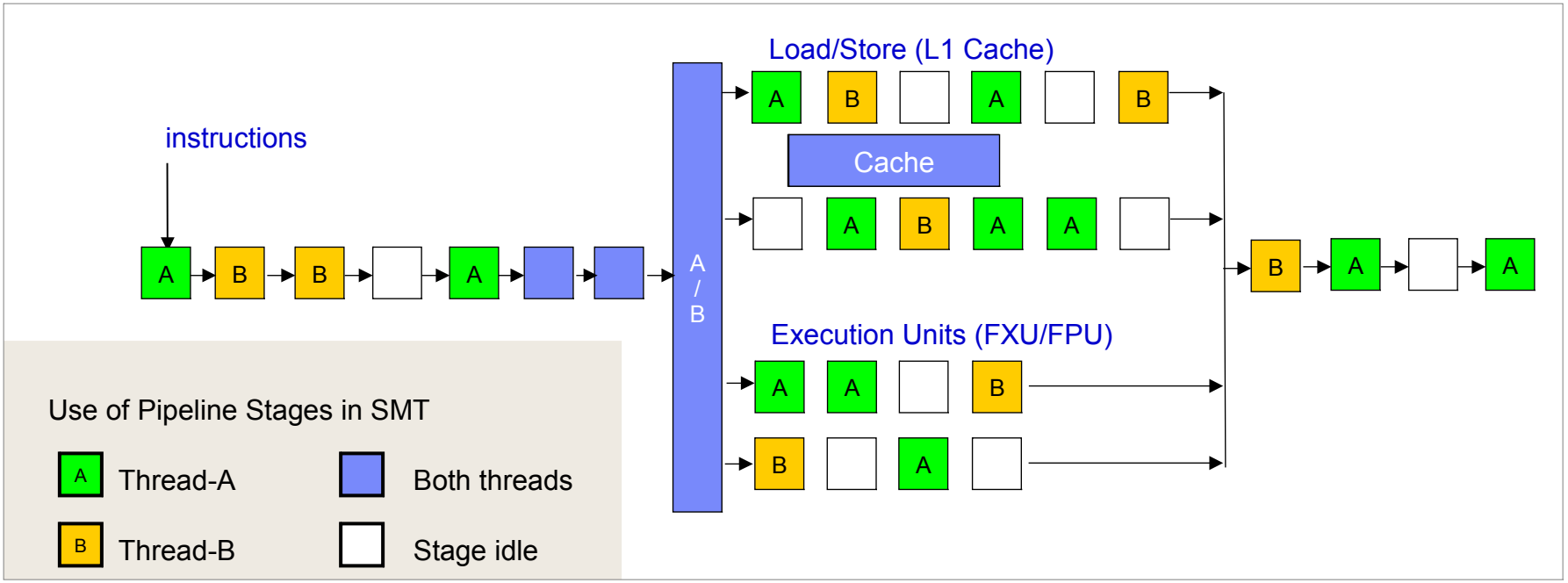
Agenda

- Your CPU is a supermarket
  - Clock
  - TLB
  - Cache
  - µOps
  - Superscalar
  - Instruction grouping
  - Out of Order execution
  - Symmetric MultiProcessing
    - Shared resources
    - Locality
  - Predictions
  - Simultaneous MultiThreading
  - Specialty Engines and Spares
  - Full CEC
  - Virtualization

- SMT for Linux on z Systems
  - Once more with theory
  - SMT resources
  - Additional complexities
    - Hypervisor scheduling complexities
    - ETR / ITR – be careful with calculations
    - Additional memory footprint
    - Additional overhead
    - Utilization 50, 100, 200%?
    - Scheduling spill and fill
  - Appearance in a Linux System
  - Benchmark examples
    - The good
    - The bad
    - And the ugly

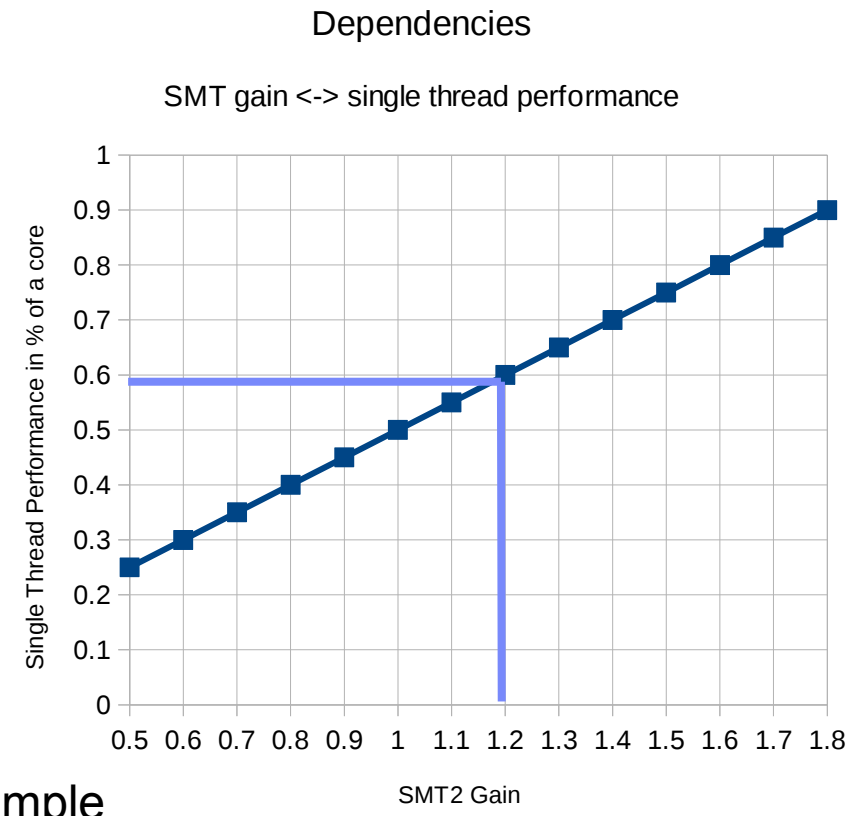
Simultaneous Multithreading – Once more with theory

- **Simultaneous Multithreading (SMT) technology**
  - Multiple programs (software threads) run on the same processor core
  - More efficient use of the core hardware
- **Active threads share core resources**
  - In space: data and instruction caches, TLBs, branch history tables, etc.
  - In time: pipeline slots, execution units, address translator, etc.
- **Typically increases overall throughput per core when SMT is active**
  - Amount that increase, varies widely with workload
  - Each thread runs slower than on a single-thread core



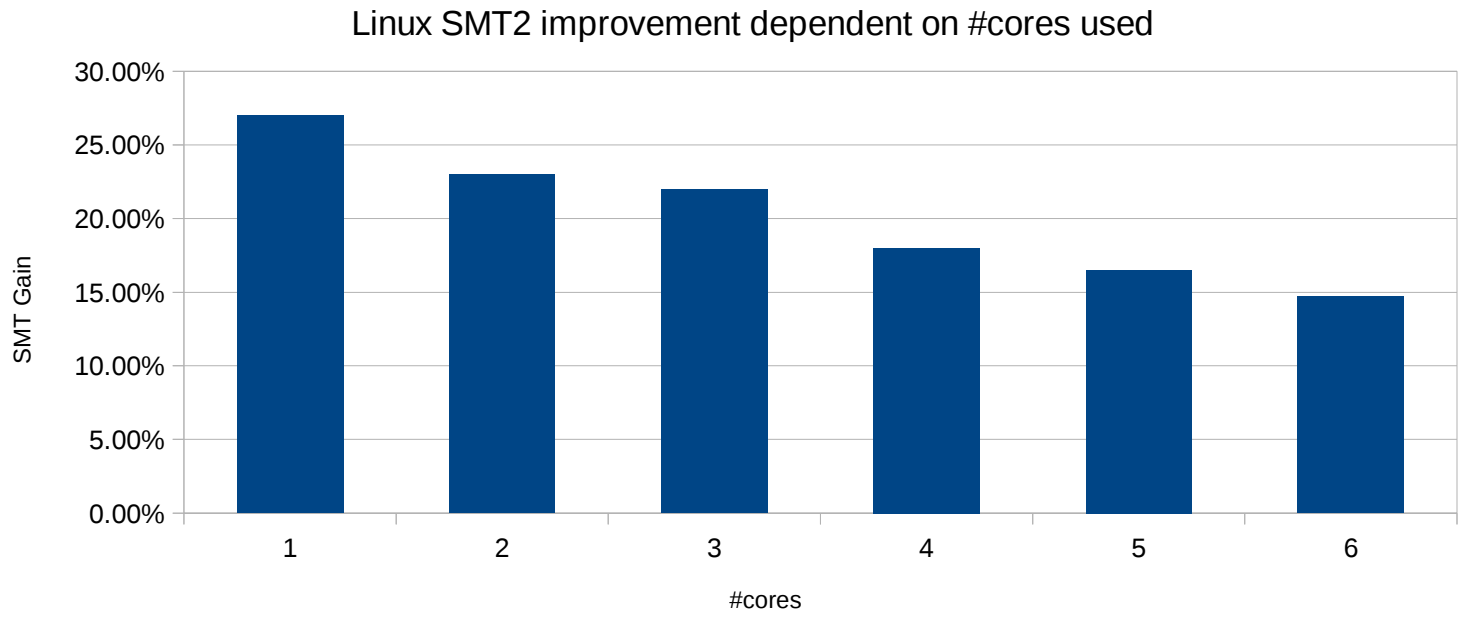
SMT2 implies that each logical CPU is slower

- Evaluate your workload
  - Do you have a single thread speed dependency?
  - logwriter process?
  - I/O processing?
- Mixed workloads?
  - Per Hypervisor config in z/VM
  - Per OS for Linux in LPAR
- All averages – workloads vary
  - SMT gain 1.2 is just an individual example
  - Gain/Loss doesn't have to be distributed equally



SMT2 gain dependent on #cores used

- Cores on a chip share resources
  - "Imbalanced" demand can deplete a resource when using more cores → can be a scaling bottleneck
  - Can happen earlier with SMT
  -



  - This is one example. There is a lot of variability with regard to workload benefit

SMT2 requires more virtual/logical CPUs

- Reduced single thread capacity

  → more CPUs required for same capacity
- More CPUs

  → negative SMP n-way effect
- n-way scalability very important
- Revisit your #virtual CPU sizing
- Measure before you deploy

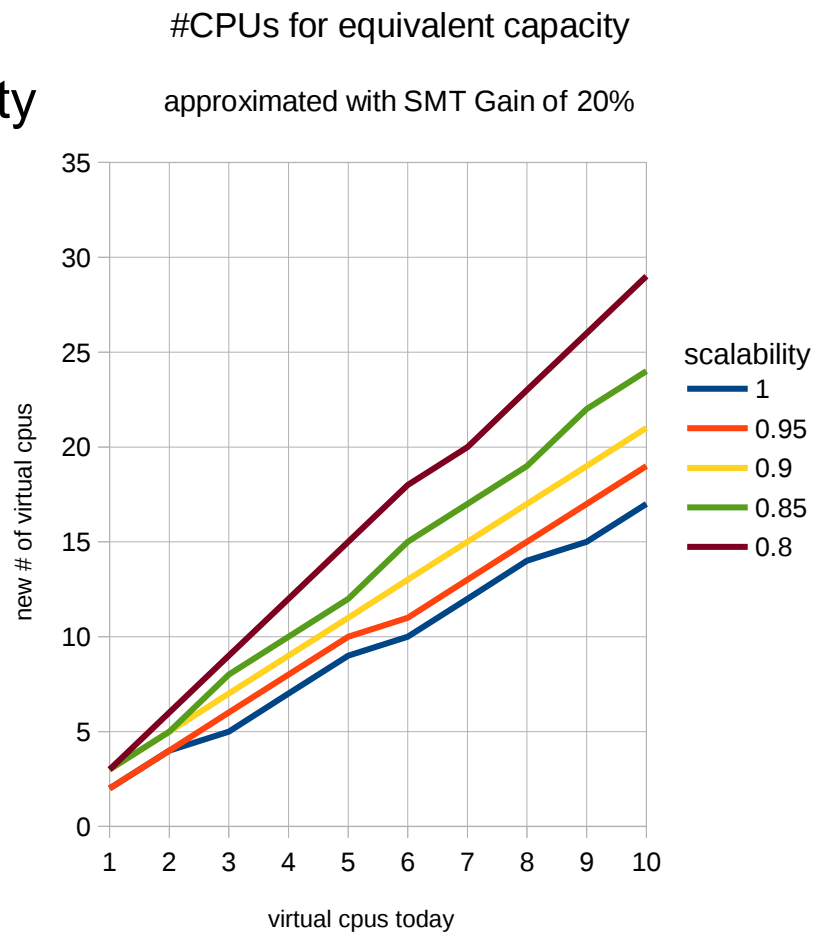
Approximation:

K_0 = *OldSizing / STcapacity*

K_n = *((*K_{n-1}*/*scalability) –* K_{n-1}*) / STcapacity*

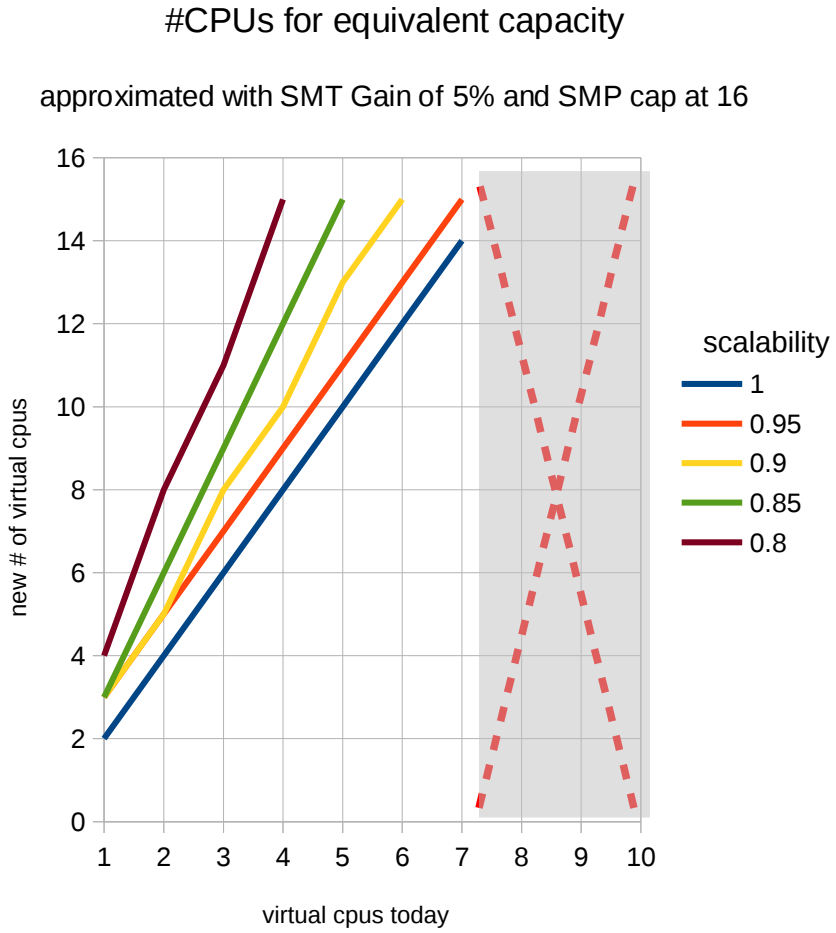
NewSizing = round-up ($\Sigma(K_0 .. K_\infty)$ *)*

Workloads differences affect this so much, that better formulas aren't more exact



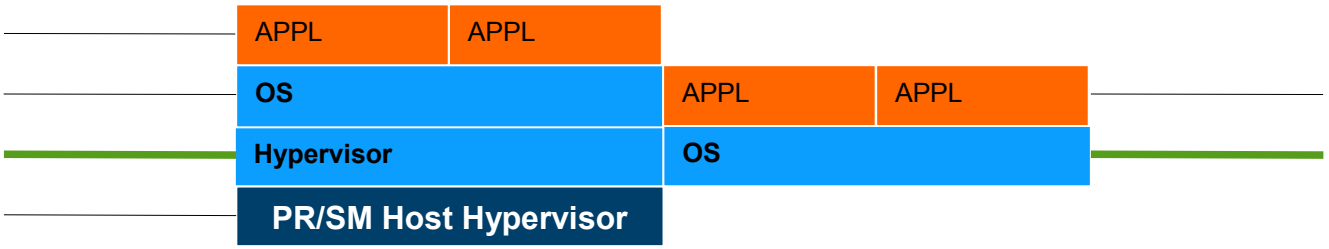
SMT2 requires more virtual/logical CPUs (cont.)

- Runaway effect possible
  - Workloads with less SMT gain
  - Workloads with SMP cap
- n-way scalability very important



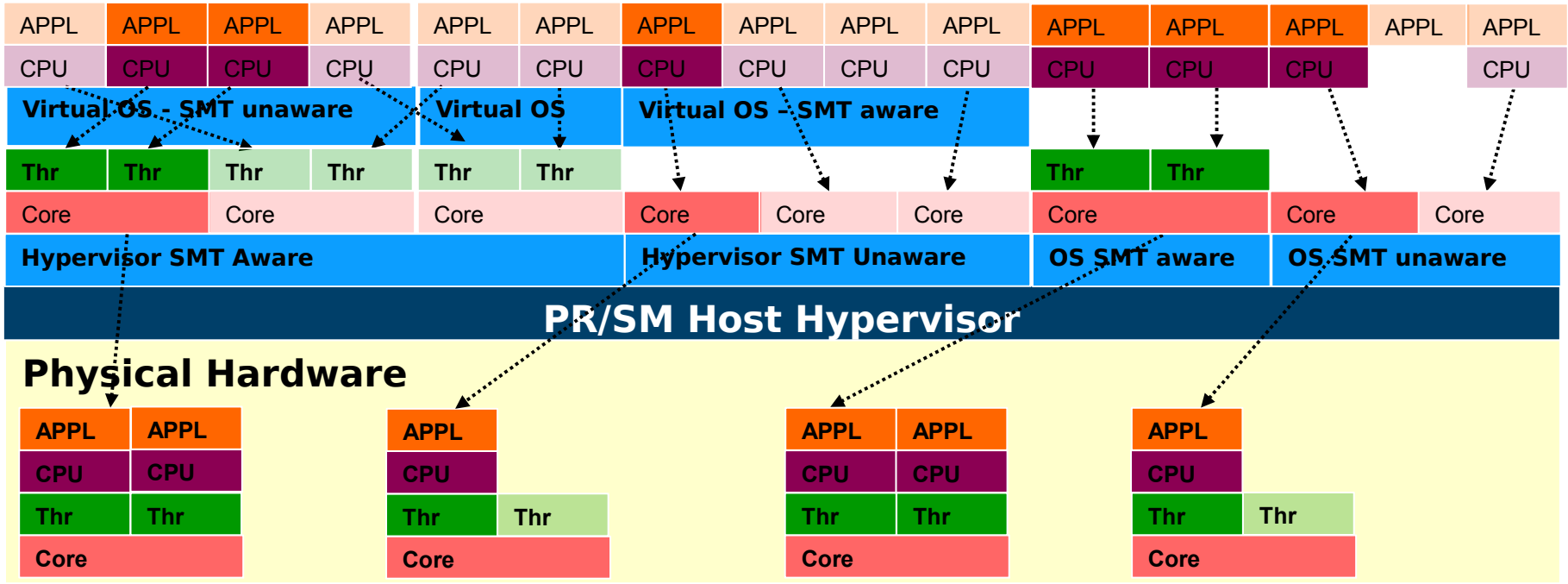
SMT - Hypervisor scheduling complexities I

- One could imagine splitting cores into threads on any level
  - When a higher level splits cores an HV exit might exit both threads
- Currently implemented in Guest 1 Supervisor via z/VM and Linux
  - Need to be supported in lower, but transparent to higher layers
  - Enough complexity already
- Hypervisor supporting SMT can either
  - Control, manage and pass whole cores (all threads) (PR/SM)
  - Dispatch threads as vcpu transparent to OSes and their applications (z/VM)
- Operating systems supporting SMT
  - Dispatch threads as logical CPUs transparent to applications (Linux)



SMT - Hypervisor scheduling complexities II

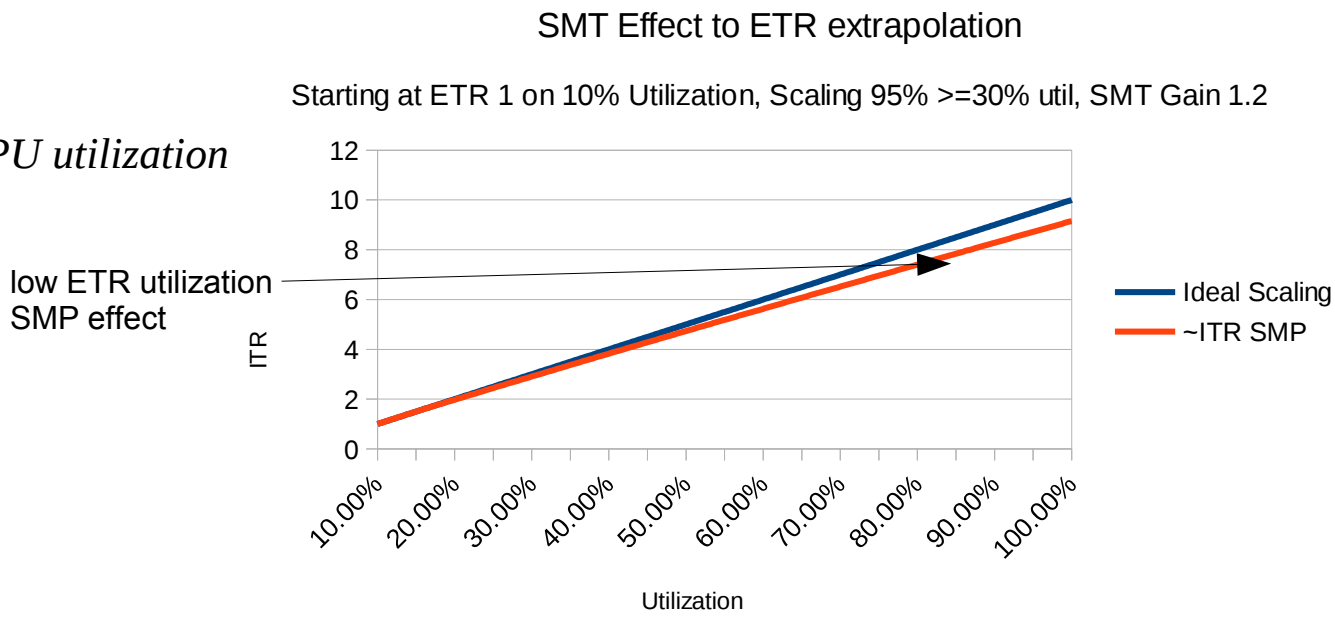
- Each level does dispatching
  - SMT aware levels split cores and does dispatch on threads
  - SMT unaware levels dispatch on their CPUs (whatever they really are)
  - More complexity
    - Which virtual logical CPU will currently be backed with a real core/thread?
    - Cross effects between threads are much higher than between cores



ETR / ITR – be careful with calculations

- Example with SMT2 enabled
  - ETR at 50% utilization of the logical CPUs
  - Scheduler put them on different cores
  - Throughput is roughly equivalent to what you get with SMT1
    → ~5/6 of total capacity
  - ETR taken at low utilization doesn't consider shared resources well
  - Normal calculation ITR = ETR / Utilization
    - Assumes a SMT gain factor of 100%
    - Wrong result

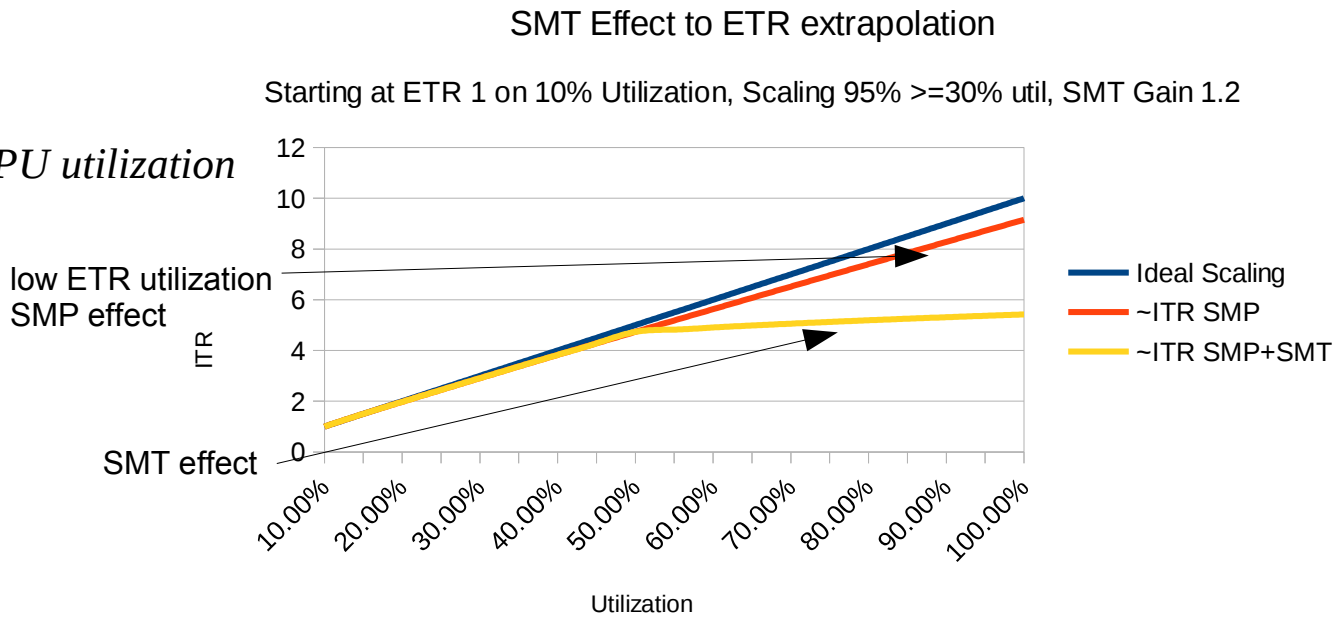
$$ETR = \frac{Transactions}{Elapsed\,time} = ITR * CPU\ utilization$$



ETR / ITR – be careful with calculations

- Example with SMT2 enabled
  - ETR at 50% utilization of the logical CPUs
  - Scheduler put them on different cores
  - Throughput is roughly equivalent to what you get with SMT1
    → ~5/6 of total capacity
  - ETR taken at low utilization doesn't consider shared resources well
  - Normal calculation ITR = ETR / Utilization
    - Assumes a SMT gain factor of 100%
    - Wrong result

$$ETR = \frac{Transactions}{Elapsed\ time} = ITR * CPU\ utilization$$



Additional complexities due to SMT

- Additional memory footprint
  - Some Structures are required for every logical CPU
- Additional overhead
  - Some tasks are carried out for every logical CPU

- Scheduling – (not only) spill and fill
  - Spill to each core with 1 thread on each (max performance)
  - Then fill up 2nd threads (capacity)
  - Many trade-offs to be taken
    - Which core to burden with extra threads
    - Avoid hitting those with single thread performance dependencies
    - Heavy Inter Process Tasks (IPC) tasks could be better filled on one core before spilled due to locality
    - Spill and Fill has to be balanced against topology constraints

Appearance of SMT in Linux I

- Controlled via Kernel parameter
  - default enabled
  - option “nosmt” disables the functionality

- Looks and behaves like a normal CPU

```
cat /proc/cpuinfo
vendor_id       : IBM/S390
# processors    : 54
[…]
processor 0: version = 00,  identification = 016F27,  machine = 2964
processor 1: version = 00,  identification = 016F27,  machine = 2964
processor 2: version = 00,  identification = 016F27,  machine = 2964
processor 3: version = 00,  identification = 016F27,  machine = 2964
[…]
```

```
lscpu -e
CPU NODE BOOK SOCKET CORE L1d:L1i:L2d:L2i ONLINE CONFIGURED POLARIZATION ADDRESS
0   0    0    0      0    0:0:0:0         yes    yes        horizontal   0
1   0    0    0      0    1:1:1:1         yes    yes        horizontal   1
2   0    0    0      1    2:2:2:2         yes    yes        horizontal   2
3   0    0    0      1    3:3:3:3         yes    yes        horizontal   3
[...]
```

Appearance of SMT in Linux II

- Looks and behaves like a normal CPU

```
top - 14:46:35 up 15 min,  2 users,  load average: 2.83, 0.66, 0.25
Tasks: 350 total,  22 running, 328 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  :  0.0 us, 36.7 sy,  0.0 ni, 61.3 id,  0.0 wa,  0.0 hi,  0.0 si,  2.0 st
%Cpu2  :  0.0 us, 93.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  7.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

- Extra Scheduling domain for threads

```
cat /proc/sys/kernel/sched_domain/cpu*/domain*/name  | sort | uniq
BOOK (= Node on z13)
MC   (= Chip on z13)
SMT  (= Threads)
```

Appearance of SMT in Linux III

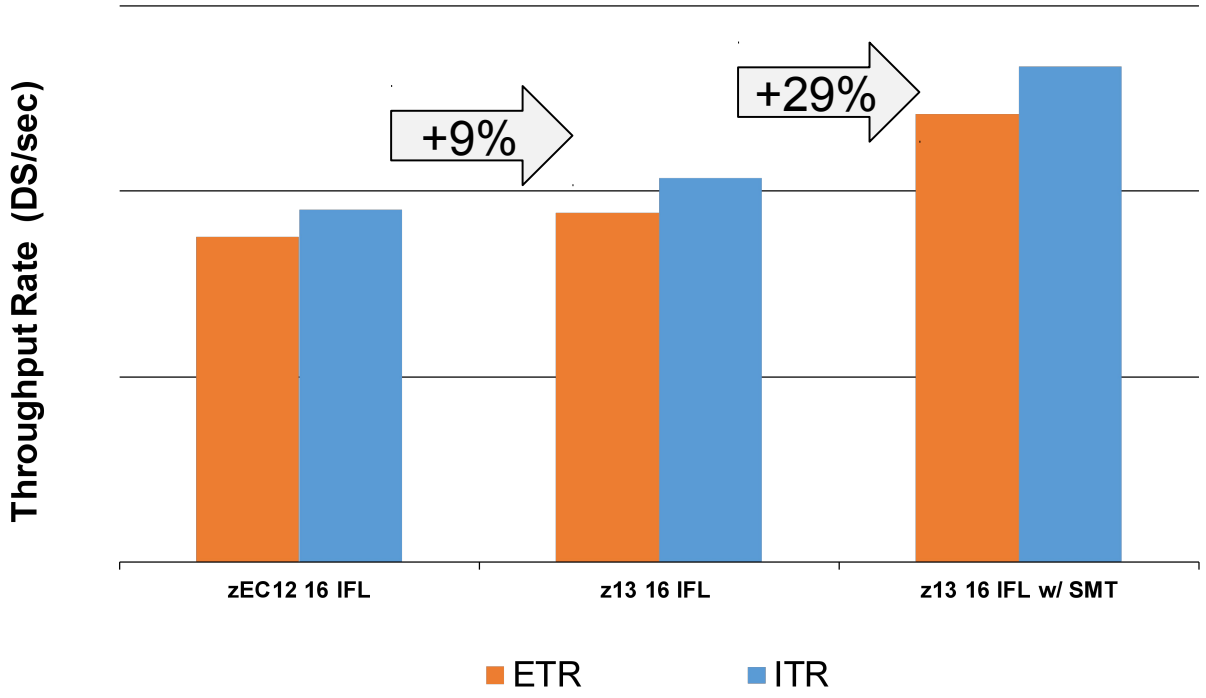
- Utilization 50, 100, 200%?
  - Each thread can be utilized 0 -100% of the time
  - Accounting based purely on Linux scheduler
  - So in a SMT-2 system
    - 50% overall utilization means ~all cores are used once
    - 100% overall utilization means all threads are used
    - 100% isn't 2 times as fast as 50%
    - There is no 200% of cores used (e.g. diag204)
  - Core based utilization information differs
    - Linux 0%→50% via Diag 204 in hyptop shows 0%→100%
    - Linux 50%→100% via Diag 204 in hyptop always shows 100%
    - Depends on the scheduler filling all cores before spilling to threads

Appearance of SMT in Linux IV

- Utilization 50, 100, 200%?
  - Generic interfaces don't include information about threading efficiency
  - In the architecture are CPUMF based fields for calculations
  - SMT weighted utilization available in taskstats interface
    https://www.kernel.org/doc/Documentation/accounting/taskstats.txt
  - Experience with SMT on x86 and Power has proven them misleading
  - Currently no common tool consumes this information

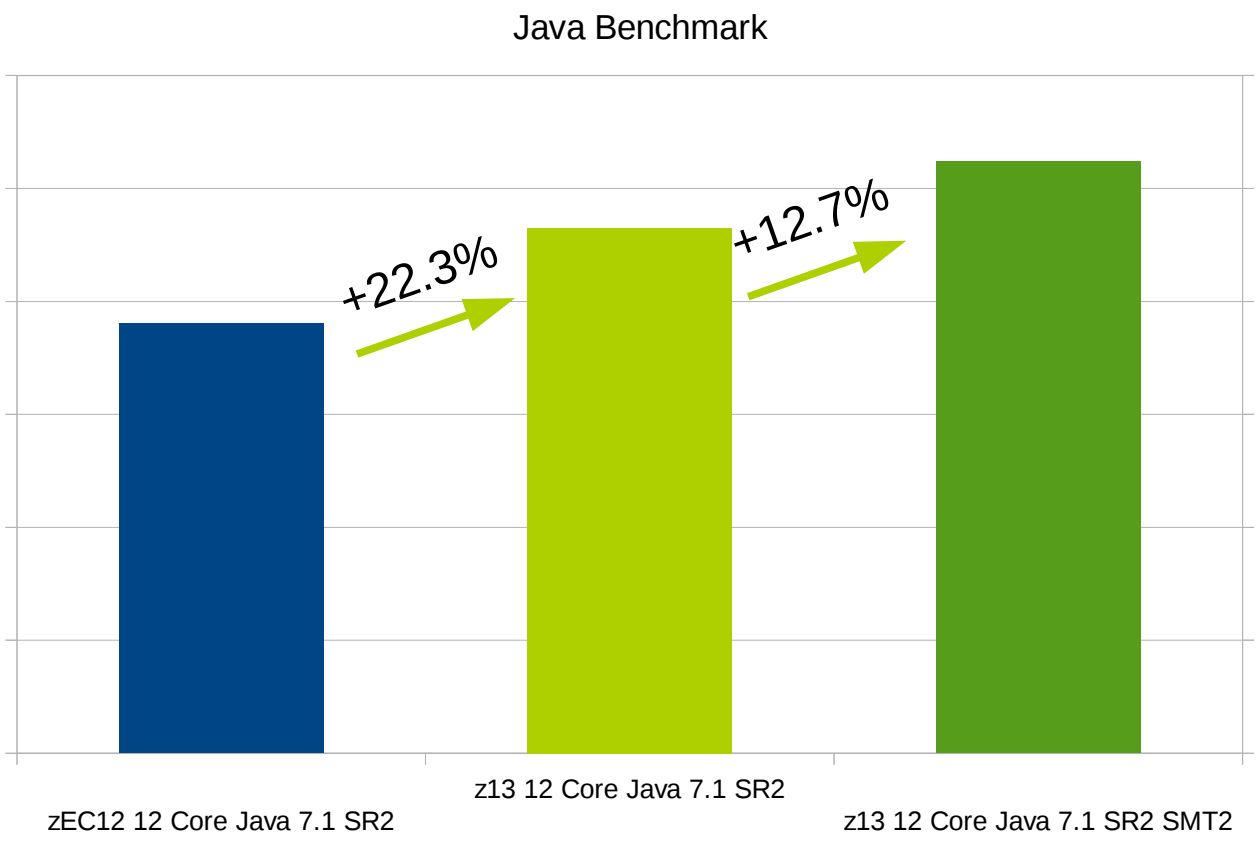
SMT real world example – the good

- SAP Workload
  - 2CPs, 2 zIIPs, DB server
  - 16 IFL App Server
  - SMT2 with z/VM
  - Overall +41% ITR



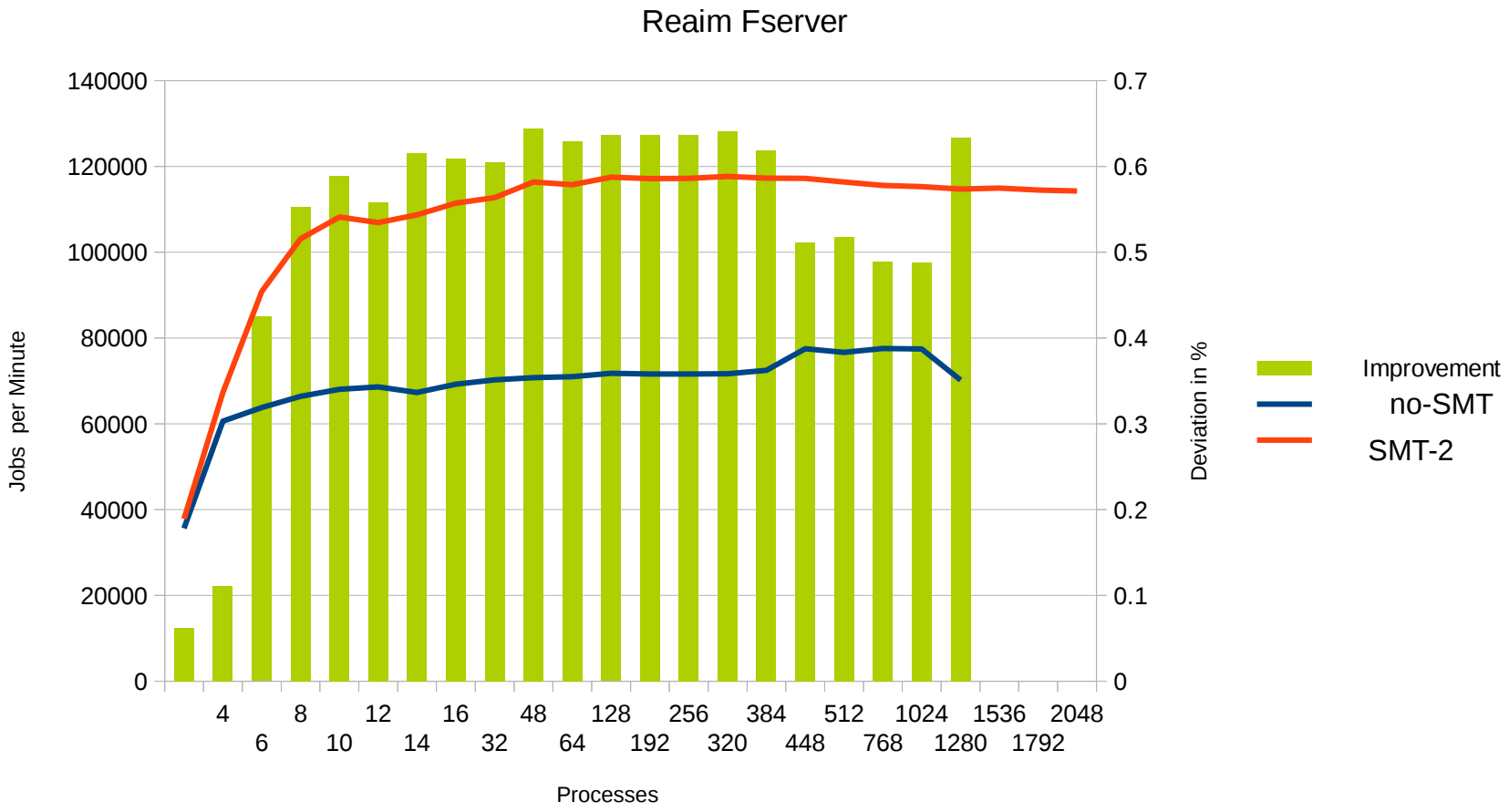
SMT real world example – the good

- Java Benchmark
  - 12 Cores – 3 (H) / 2 (S) Chips
  - Staying inside one Node/Book
  - SMT2 with Linux
  - overall ~40% ITR



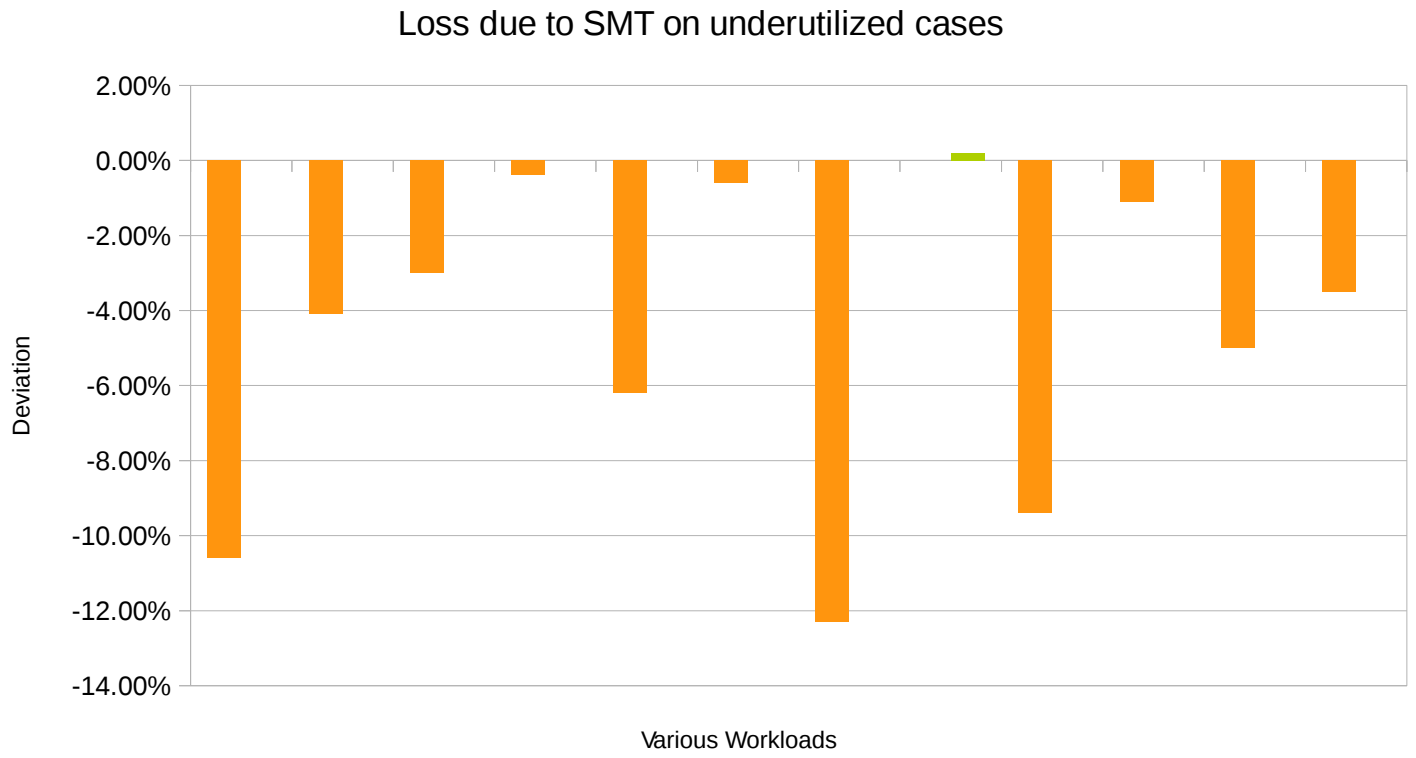
SMT real world example – the good

- Some cases just win everywhere



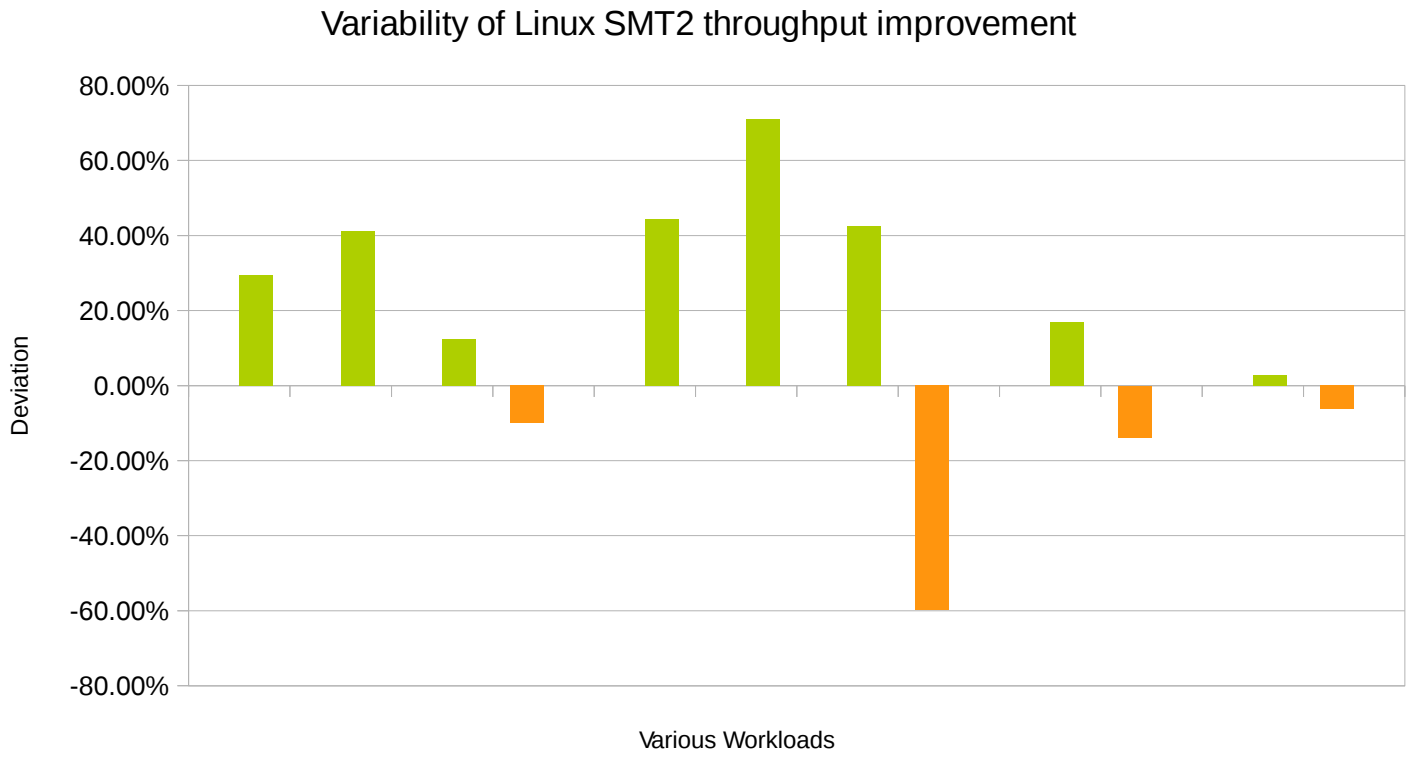
SMT real world example – the bad

- Some other cases still need some optimizations which are work in progress



SMT real world example – the ugly

- The truth is, you have to evaluate YOUR workload



Questions

- Further information is at
  - Linux on System z – Tuning hints and tips
    http://www.ibm.com/developerworks/linux/linux390/perf/index.html
  - Live Virtual Classes for z/VM and Linux
    http://www.vm.ibm.com/education/lvc/

Mario Held
Linux on z Systems
Performance Evaluation

Research & Development
Schoenaicher Strasse 220
71032 Boeblingen, Germany

mario.held@de.ibm.com