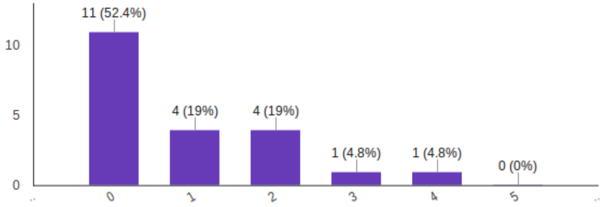
CS 6354: Memory Hierarchy I

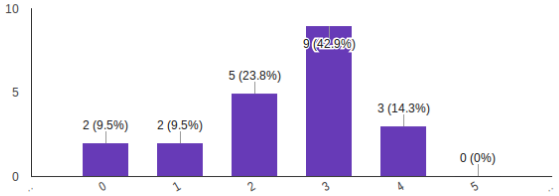
29 August 2016

Survey results

set-associative caches (21 responses)



virtual memory (21 responses)



Processor/Memory Gap

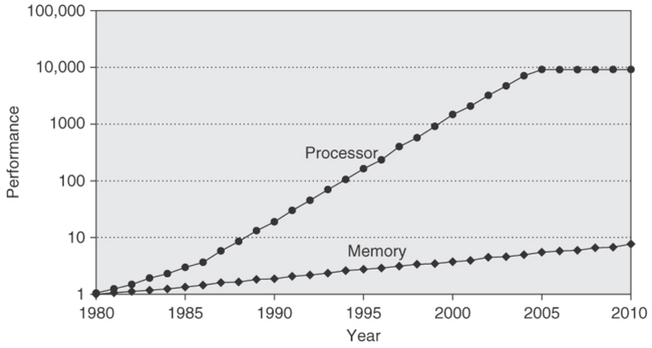


Figure 2.2 Starting with 1980 performance as a baseline, the gap in performance, measured as the difference in the time between processor memory requests (for a single processor or core) and the latency of a DRAM access, is plotted over time. Note that the vertical axis must be on a logarithmic scale to record the size of the processor–DRAM performance gap. The memory baseline is 64 KB DRAM in 1980, with a 1.07 per year performance improvement in latency (see Figure 2.13 on page 99). The processor line assumes a 1.25 improvement per year until 1986, a 1.52 improvement until 2000, a 1.20 improvement between 2000 and 2005, and no change in processor performance (on a per-core basis) between 2005 and 2010; see Figure 1.1 in Chapter 1.

Figure: H&P Ch. 2

Variety in memory technologies

SRAM

approx. 4–6 transitors/bit
optimized for speed

DRAM

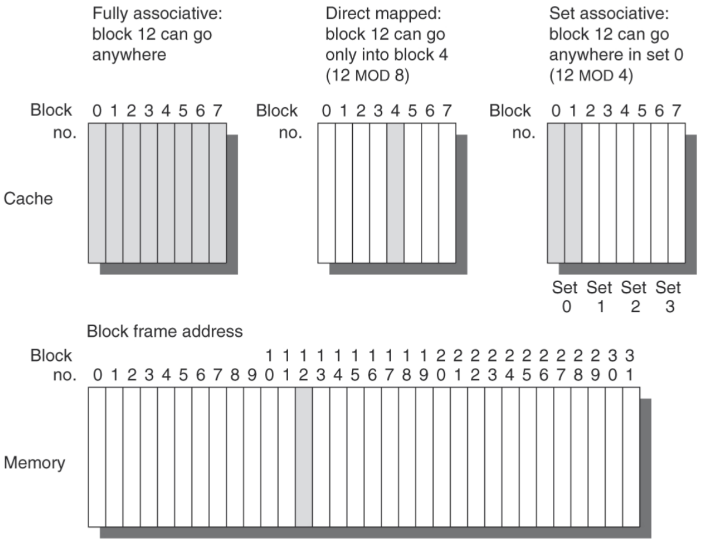
approx. 1 transitor + capacitor/bit
optimized for density

...

Also smaller $\implies$ faster

Goal: best performance and best capacity

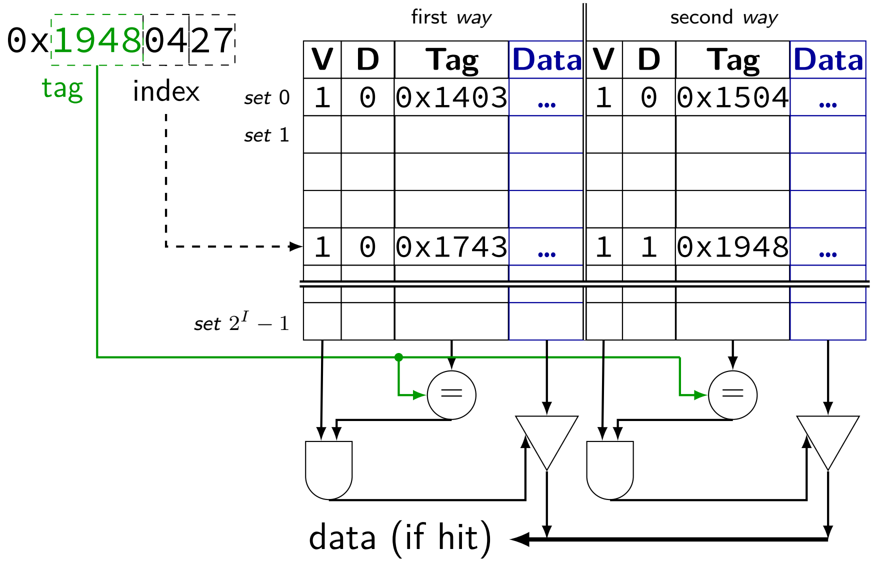
Associativity



Fully associative: block 12 can go anywhere
Direct mapped: block 12 can go only into block 4 (12 MOD 8)
Set associative: block 12 can go anywhere in set 0 (12 MOD 4)
Block no. 0 1 2 3 4 5 6 7
Cache
Set 0 Set 1 Set 2 Set 3
Block frame address
Block no. 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Memory

Figure: H&P App. B

Cache Operation



Cache Flowchart

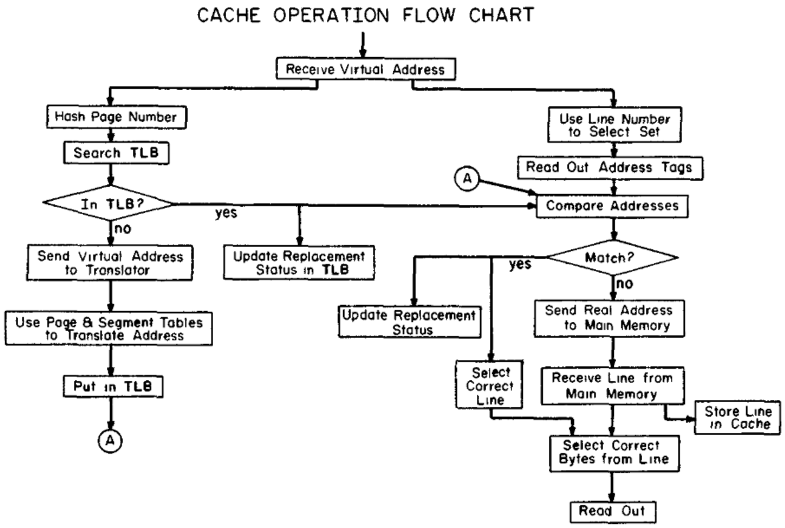
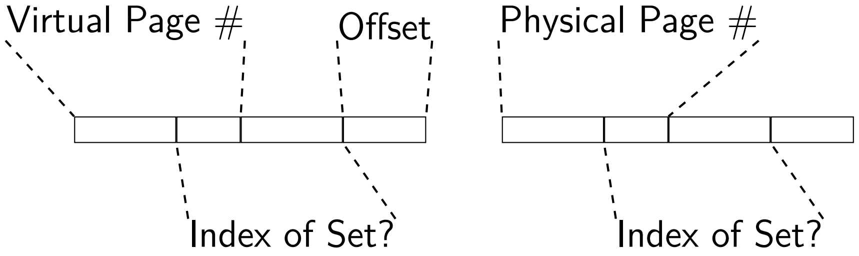
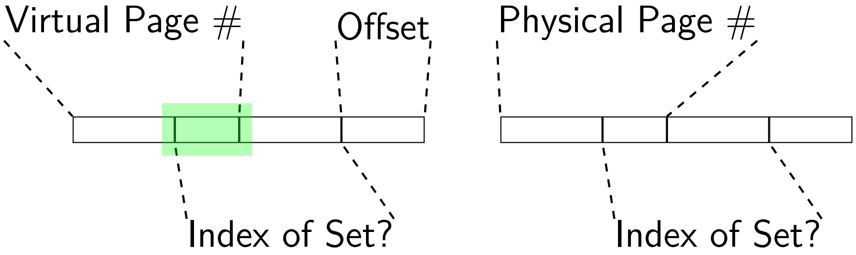

CACHE OPERATION FLOW CHART
Receive Virtual Address
Hash Page Number
Search TLB
In TLB?
yes
no
Send Virtual Address to Translator
Use Page & Segment Tables to Translate Address
Put in TLB
A
Update Replacement Status in TLB
Use Line Number to Select Set
Read Out Address Tags
A
Compare Addresses
Match?
yes
no
Update Replacement Status
Select Correct Line
Send Real Address to Main Memory
Receive Line from Main Memory
Store Line in Cache
Select Correct Bytes from Line
Read Out

Figure 3. Cache operation flow chart.

Virtual and Physical

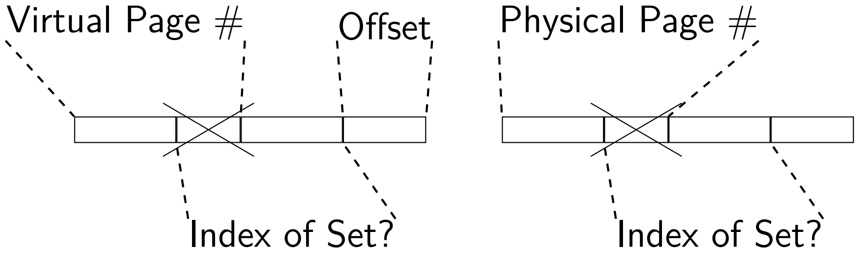


Virtual and Physical



Cache has virtual indexes?

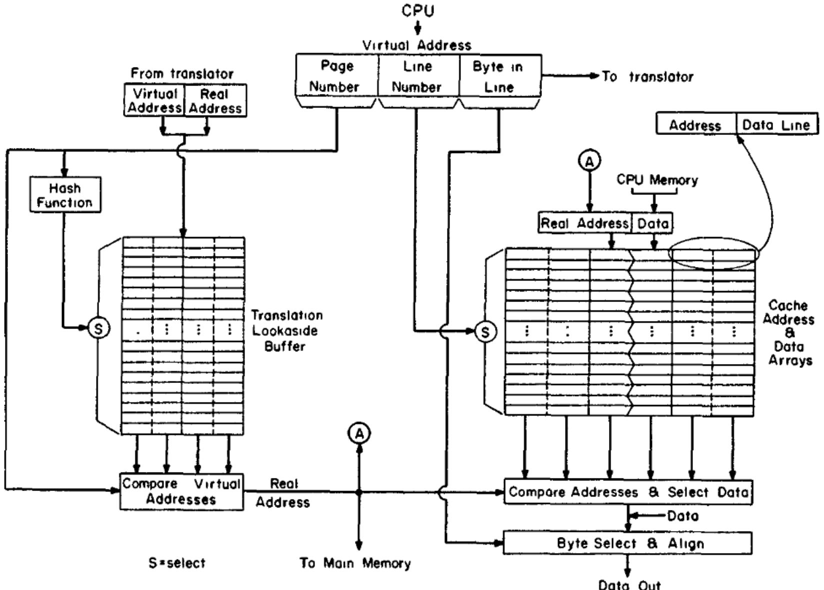
Virtual and Physical



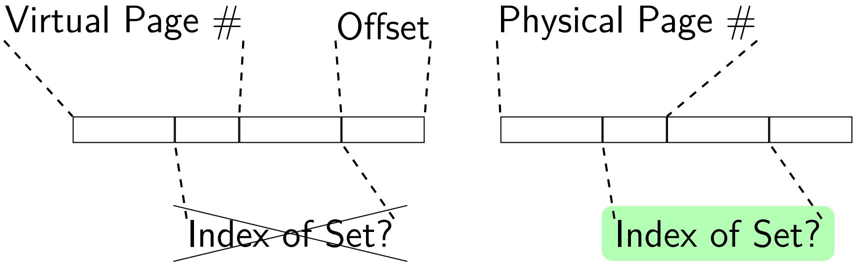
Cache has virtual indexes?

Solution #1: Disallow overlap

Cache and TLB design



Virtual and Physical



Cache has virtual indexes?

Solution #1: Disallow overlap

Solution #2: Translate first

L1: no overlap + L2: translate first

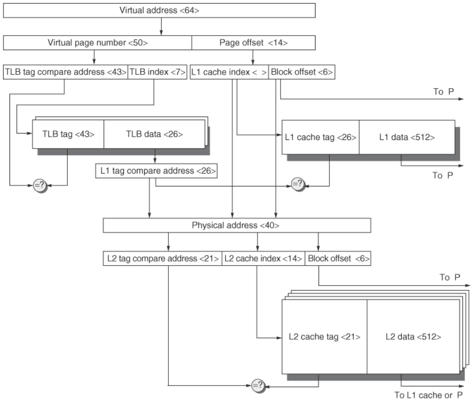
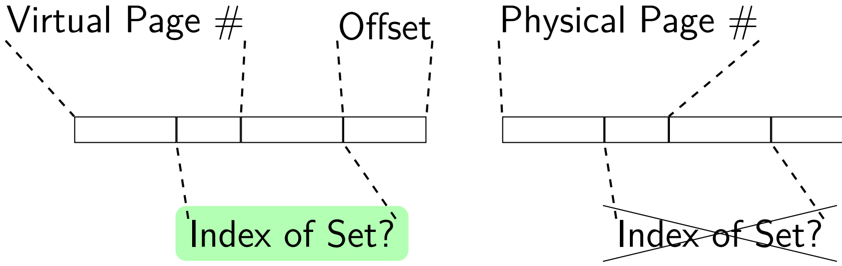


Figure B.17 The overall picture of a hypothetical memory hierarchy going from virtual address to L2 cache access. The page size is 16 KB. The TLB is two-way set associative with 256 entries. The L1 cache is a direct-mapped 16 KB, and the L2 cache is a four-way set associative with a total of 4 MB. Both use 64-byte blocks. The virtual address is 64 bits and the physical address is 40 bits.

Virtual and Physical



Cache has virtual indexes?

Solution #1: Disallow overlap

Solution #2: Translate first

Solution #3: Allow virtual indexes (with overlap)

Virtually Indexed Caches Issues

Virtually Indexed Caches Issues

do tags store physical or virtual addresses?

Virtually Indexed Caches Issues

do tags store physical or virtual addresses?

what about page table changes?

Virtually Indexed Caches Issues

do tags store physical or virtual addresses?

what about page table changes?

two virtual addresses (aliasing) for same physical address?

Types of Cache Misses

compulsory — first access

capacity — not enough space

conflict — enough space, but in wrong sets

(coherency — synchronizing between caches)

average memory access time (AMAT)

$$\text{AMAT} = \text{hit time} + \text{miss rate} \times \text{miss penalty}$$

Smith paper summary

Simulation study

Whole bunch of cache parameters

Optimize by applying simulation

The Smith paper's simulator

Input?

The Smith paper's simulator

Input?

What's missing?

miss rate versus AMAT

Every optimization has a complexity cost!

Can you figure that out from the paper we read?

miss rate or AMAT versus program performance

How many memory accesses per instruction?

Evenly distributed?

What if all instructions have lots of computation?

What systems that can ‘hide’ misses with speculation?

write stratagies

write-through — write on change

write-back — write on cache eviction

also known as copy-back

replacement policies

random

fifo

LRU approximations

if write-back:

write-allocate — bring into cache on write

no write-allocate — write-through if not yet in cache

write buffers

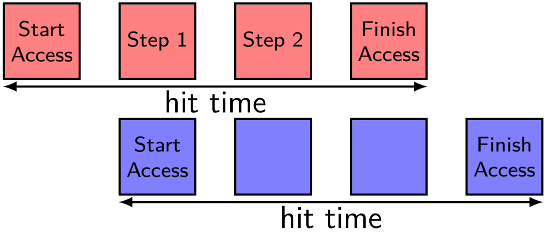
data no longer in cache but not yet in memory

need to be checked on reads

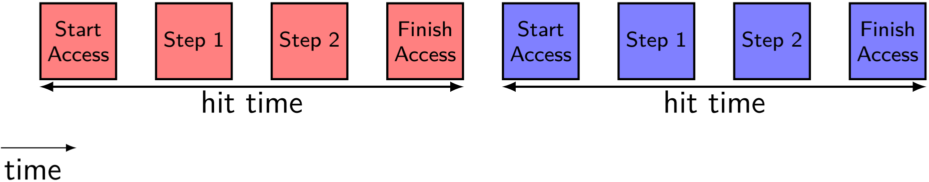
pipelined caches

Pipelined: start one cache access per cycle

Same latency (hit time), higher bandwidth



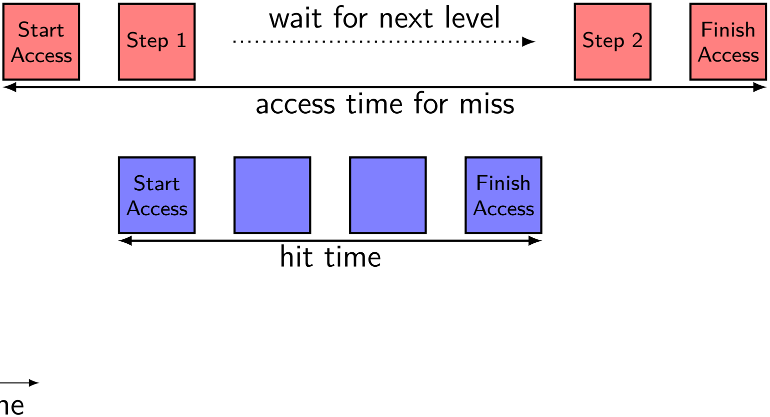
vs.



Non-blocking

Non-blocking: "Hit under miss"; complete accesses out-of-order

Best if CPU can work out-of-order



Non-blocking improvement

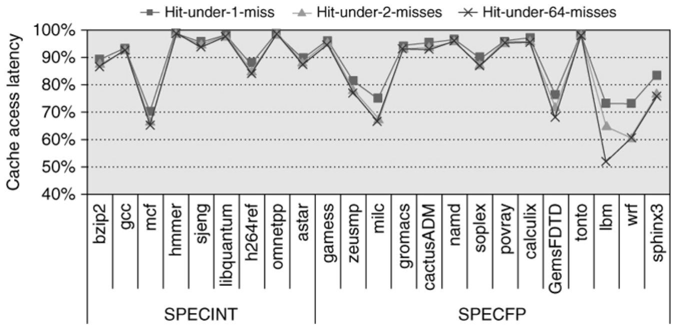


Figure 2.5 The effectiveness of a nonblocking cache is evaluated by allowing 1, 2, or 64 hits under a cache miss with 9 SPECINT (on the left) and 9 SPECFP (on the right) benchmarks. The data memory system modeled after the Intel i7 consists of a 32KB L1 cache with a four cycle access latency. The L2 cache (shared with instructions) is 256 KB with a 10 clock cycle access latency. The L3 is 2 MB and a 36-cycle access latency. All the caches are eight-way set associative and have a 64-byte block size. Allowing one hit under miss reduces the miss penalty by 9% for the integer benchmarks and 12.5% for the floating point. Allowing a second hit improves these results to 10% and 16%, and allowing 64 results in little additional improvement.

Cache Optimizations

Technique	Improves			
	Hit time	Miss penalty	Hit rate	Band-width
Increase block size		N	Y	
Increase cache size	N		Y	
Increase associavity	N		Y	
Multilevel caches		Y		
Prioritize reads over writes		Y		
Virtual-index = Physical	Y			
Pipelined cache accesses				Y
Non-blocking caches		Y		Y
Prefetching		Y	Y	
Way-prediction	Y			

+ complexity costs

(adapted from tables in H&P B and H&P 2.2)

Cache Optimizations

Technique	Improves Hit time	Miss penalty	Hit rate	Band-width
Increase block size		N	Y	
Increase cache size	N		Y	
Increase associavity	N		Y	
Multilevel caches		Y		
Prioritize reads over writes		Y		
Virtual-index = Physical	Y			
Pipelined cache accesses				Y
Non-blocking caches		Y		Y
Prefetching		Y	Y	
Way-prediction	Y			

+ complexity costs

(adapted from tables in H&P B and H&P 2.2)

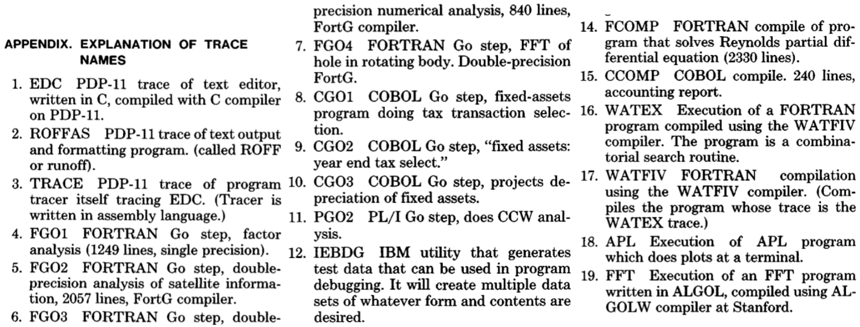
Cache Optimizations

Technique	Improves Hit time	 Miss penalty	 Hit rate	 Bandwidth
Increase block size		N	Y	
Increase cache size	N		Y	
Increase associavity	N		Y	
Multilevel caches		Y		
Prioritize reads over writes		Y		
Virtual-index = Physical	Y			
Pipelined cache accesses				Y
Non-blocking caches		Y		Y
Prefetching		Y	Y	
Way-prediction	Y			

+ complexity costs

(adapted from tables in H&P B and H&P 2.2)

Choice of traces



APPENDIX. EXPLANATION OF TRACE NAMES

1. EDC PDP-11 trace of text editor, written in C, compiled with C compiler on PDP-11.
2. ROFFAS PDP-11 trace of text output and formatting program. (called ROFF or runoff).
3. TRACE PDP-11 trace of program tracer itself tracing EDC. (Tracer is written in assembly language.)
4. FGO1 FORTRAN Go step, factor analysis (1249 lines, single precision).
5. FGO2 FORTRAN Go step, double-precision analysis of satellite information, 2057 lines, FortG compiler.
6. FGO3 FORTRAN Go step, double-precision numerical analysis, 840 lines, FortG compiler.
7. FGO4 FORTRAN Go step, FFT of hole in rotating body. Double-precision FortG.
8. CGO1 COBOL Go step, fixed-assets program doing tax transaction selection.
9. CGO2 COBOL Go step, "fixed assets: year end tax select."
10. CGO3 COBOL Go step, projects depreciation of fixed assets.
11. PGO2 PL/I Go step, does CCW analysis.
12. IEBDG IBM utility that generates test data that can be used in program debugging. It will create multiple data sets of whatever form and contents are desired.
14. FCOMP FORTRAN compile of program that solves Reynolds partial differential equation (2330 lines).
15. CCOMP COBOL compile. 240 lines, accounting report.
16. WATEX Execution of a FORTRAN program compiled using the WATFIV compiler. The program is a combinatorial search routine.
17. WATFIV FORTRAN compilation using the WATFIV compiler. (Compiles the program whose trace is the WATEX trace.)
18. APL Execution of APL program which does plots at a terminal.
19. FFT Execution of an FFT program written in ALGOL, compiled using ALGOLW compiler at Stanford.

Mem hierarcht at ISCA'15–16 + MICRO'15

SPECcpu x16

> "compute-intensive"

NASA Parallel Benchmarks (NPB) x7

> "from computational fluid dynamics (CFD)"

PARSEC x3

> "multithreaded programs"

BioBench x3

> "a diverse set of bioinformatics applications"

(+ more that appeared less than 3 times + any I missed)

Modern cache evaluation tools

CACTI — "[a]n integrated cache and memory access time, cycle time, area, leakage, and dynamic power model"

gem5 — "a modular platform for computer-system architecture research"

Marss86

Graphite

ESESC

Flexus

...

S-Box Problem

```
static const u32 Te0[256] = {         // 4KB table
    0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b
    0xfff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5
    0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b
    0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76
// ...
    s0 = GETU32(in     ) ^ rk[0];
    s1 = GETU32(in +  4) ^ rk[1];
    s2 = GETU32(in +  8) ^ rk[2];
    s3 = GETU32(in + 12) ^ rk[3];
    /* round 1: */
    t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^
    // ...
```

rk — round key — TOP SECRET

S-Box Problem

```
static const u32 Te0[256] = {        // 4KB table
    0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b
    0xfff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5
    0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b
    0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76
// ...
    s0 = GETU32(in     ) ^ rk[0];
    s1 = GETU32(in +  4) ^ rk[1];
    s2 = GETU32(in +  8) ^ rk[2];
    s3 = GETU32(in + 12) ^ rk[3];
    /* round 1: */
    t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^
    // ...
```

rk — round key — TOP SECRET

S-Box Problem

```
static const u32 Te0[256] = {        // 4KB table
    0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b
    0xfff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5
    0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b
    0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76
// ...
    s0 = GETU32(in     ) ^ rk[0];
    s1 = GETU32(in +  4) ^ rk[1];
    s2 = GETU32(in +  8) ^ rk[2];
    s3 = GETU32(in + 12) ^ rk[3];
    /* round 1: */
    t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^
    // ...
```

rk — round key — TOP SECRET

S-Box Problem

```
static const u32 Te0[256] = {        // 4KB table
    0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b
    0xfff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5
    0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b
    0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76
// ...
    s0 = GETU32(in     ) ^ rk[0];
    s1 = GETU32(in +  4) ^ rk[1];
    s2 = GETU32(in +  8) ^ rk[2];
    s3 = GETU32(in + 12) ^ rk[3];
    /* round 1: */
    t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^
    // ...
```

rk — round key — TOP SECRET

S-Box layout

V	Tag	Data	V	Tag	Data

S-Box layout

	V	Tag	Data	V	Tag	Data
`Te0[0]—Te[3]`						
`Te0[4]—Te[7]`						
`Te0[8]—Te[11]`						

S-Box layout

	V	Tag	Data	V	Tag	Data
??? + `Te0[0]`—`Te[3]`						
??? + return addr. + `in[0]`—`in[4]` + `Te0[4]`—`Te[7]`						
??? + saved registers + `Te0[8]`—`Te[11]`						

S-Box layout

	V	Tag	Data	V	Tag	Data
??? + `Te0[0]`—`Te[3]`						
??? + return addr. + `in[0]`—`in[4]` + `Te0[4]`—`Te[7]`						
??? + saved registers + `Te0[8]`—`Te[11]`						

??? includes: OS data structures; other programs; ...

S-Box layout

	V	Tag	Data	V	Tag	Data
??? + Te0[0]—Te[3]						
??? + return addr. + in[0]—in[4] + Te0[4]—Te[7]						
??? + saved registers + Te0[8]—Te[11]						

??? includes: OS data structures; other programs; ...

varies depending on packet sizes, etc. — change

DJB's advice for CPU designers

“document the performance of their chips”

“adding AES support to their instruction sets”

“adding an L1-table-lookup instruction”

“allow [] different action upon return from interrupt”

DJB’s advice for AES implementors

“control the positions of [] variables in memory”

after “any uncontrolled code”: reload S-box

“disabl[ing] hyperthreading”

“incorporate … into the [] kernel”

“shift the stack” (relative to S-boxes)

make “each load” “take place in a separate cycle”

“Every new CPU poses a potential new challenge.”

Papers for Next Time

Jouppi, “Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers”, 1990

Cook et al, “A Hardware Evaluation of Cache Partitioning to Improve Utilization and Energy-Efficiency while Preserving Responsiveness”, 2013