The Nix Manycore Operating System

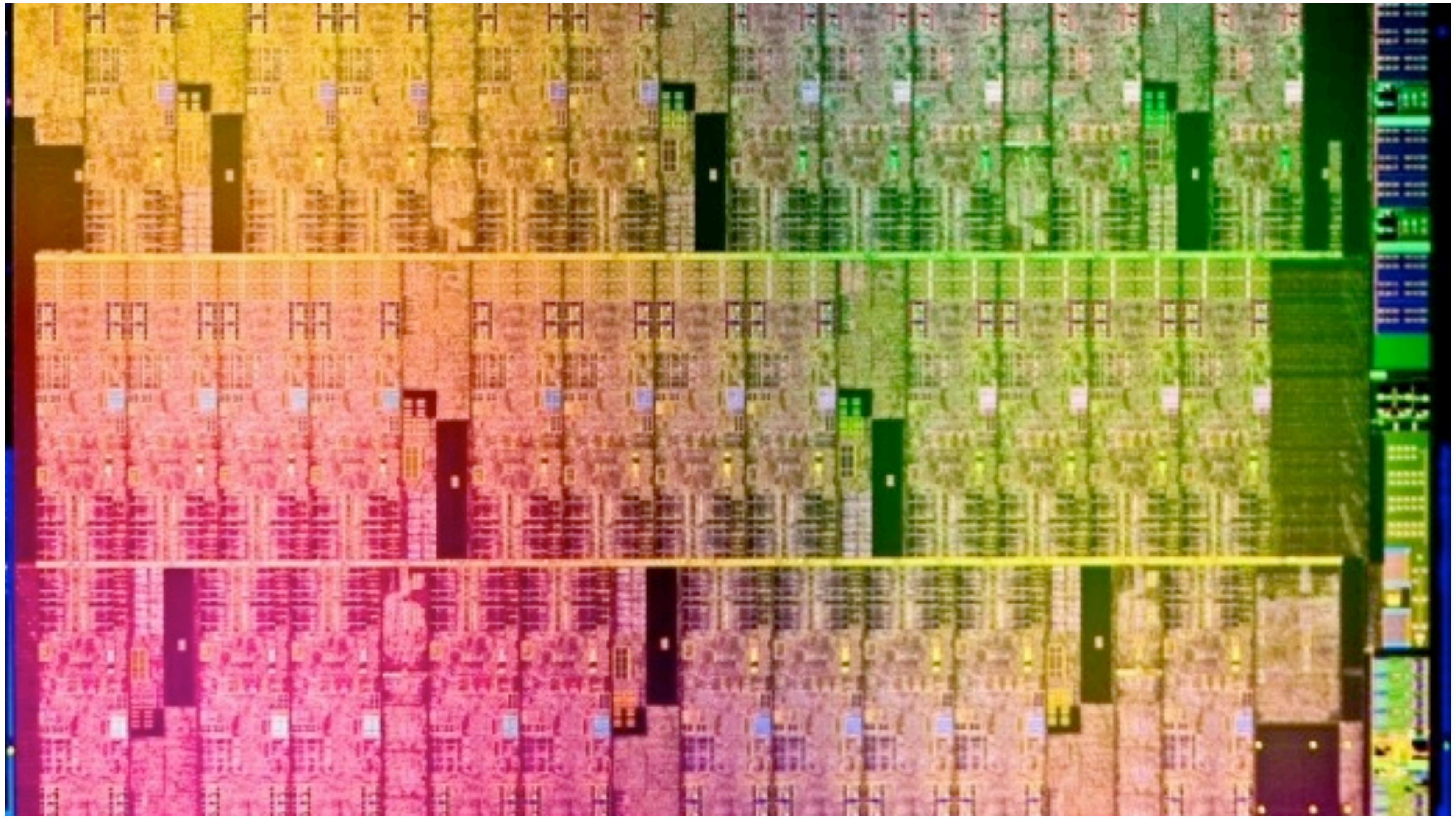
Shamelessly cribbed from a Bell Labs talk by Noah Evans
Work supported by URJC in Madrid and DOE/SC

Noah Evans
Jim McKie
Bell Labs

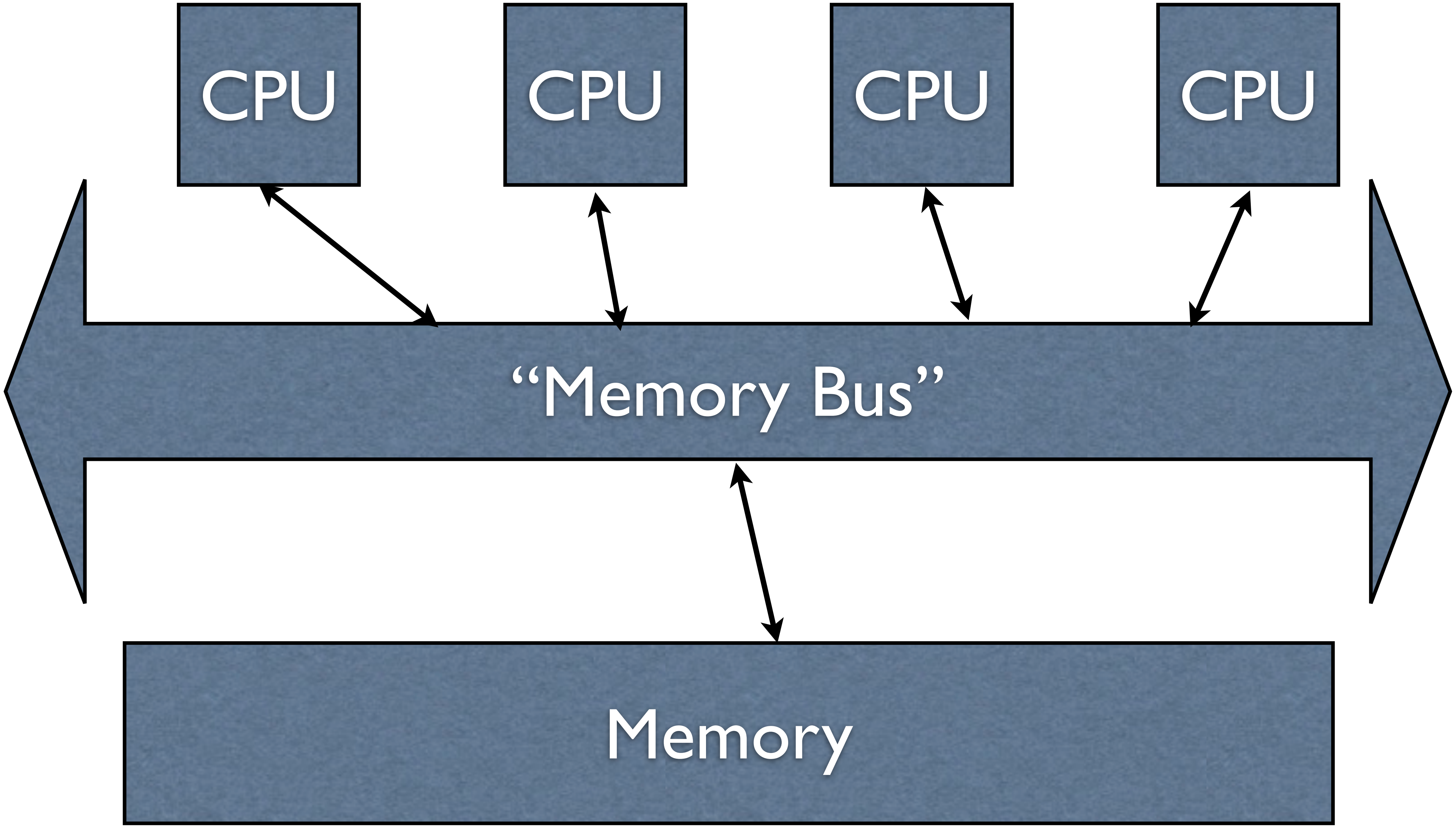
Ron Minnich
Sandia

Charles Forsyth
Vita Nuova

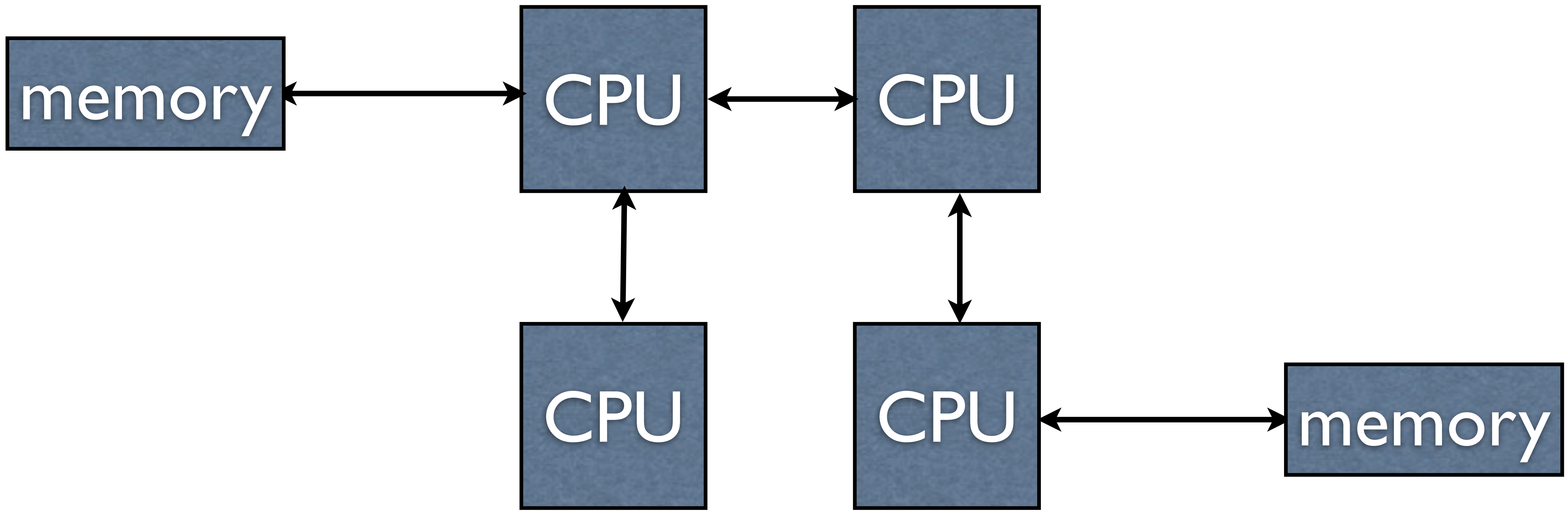
Gorka Guardiola
Fco. J. Ballesteros
Enrique Soriano
University Rey Juan Carlos



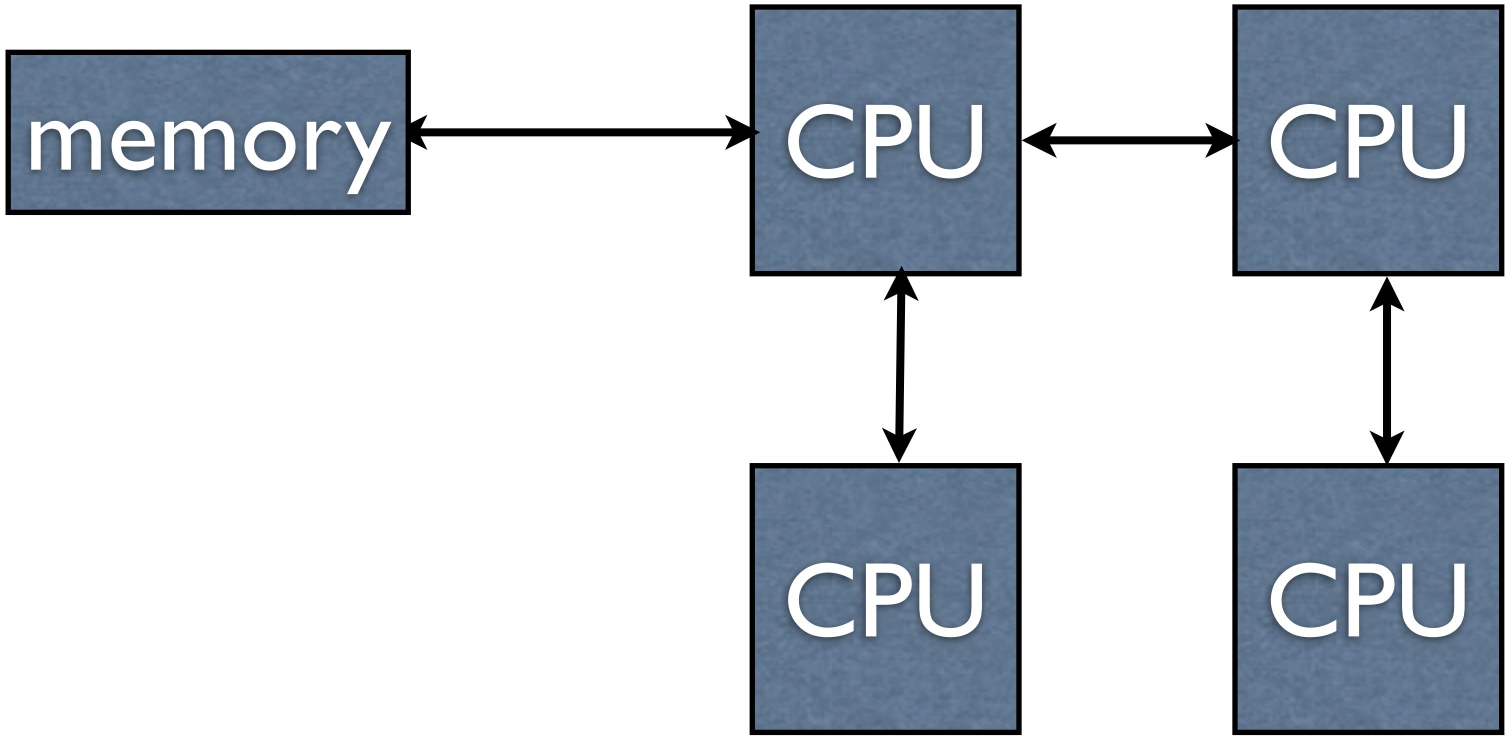
Manycore OS?



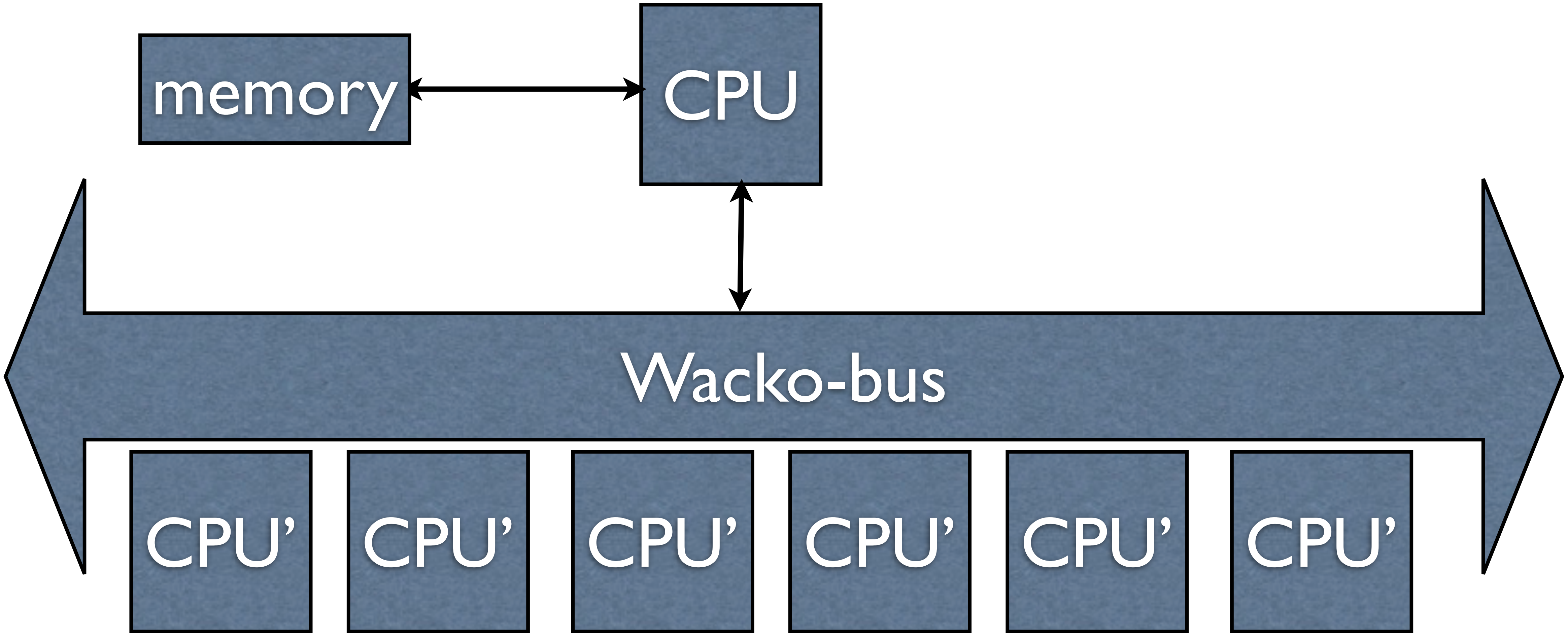
What we think we see



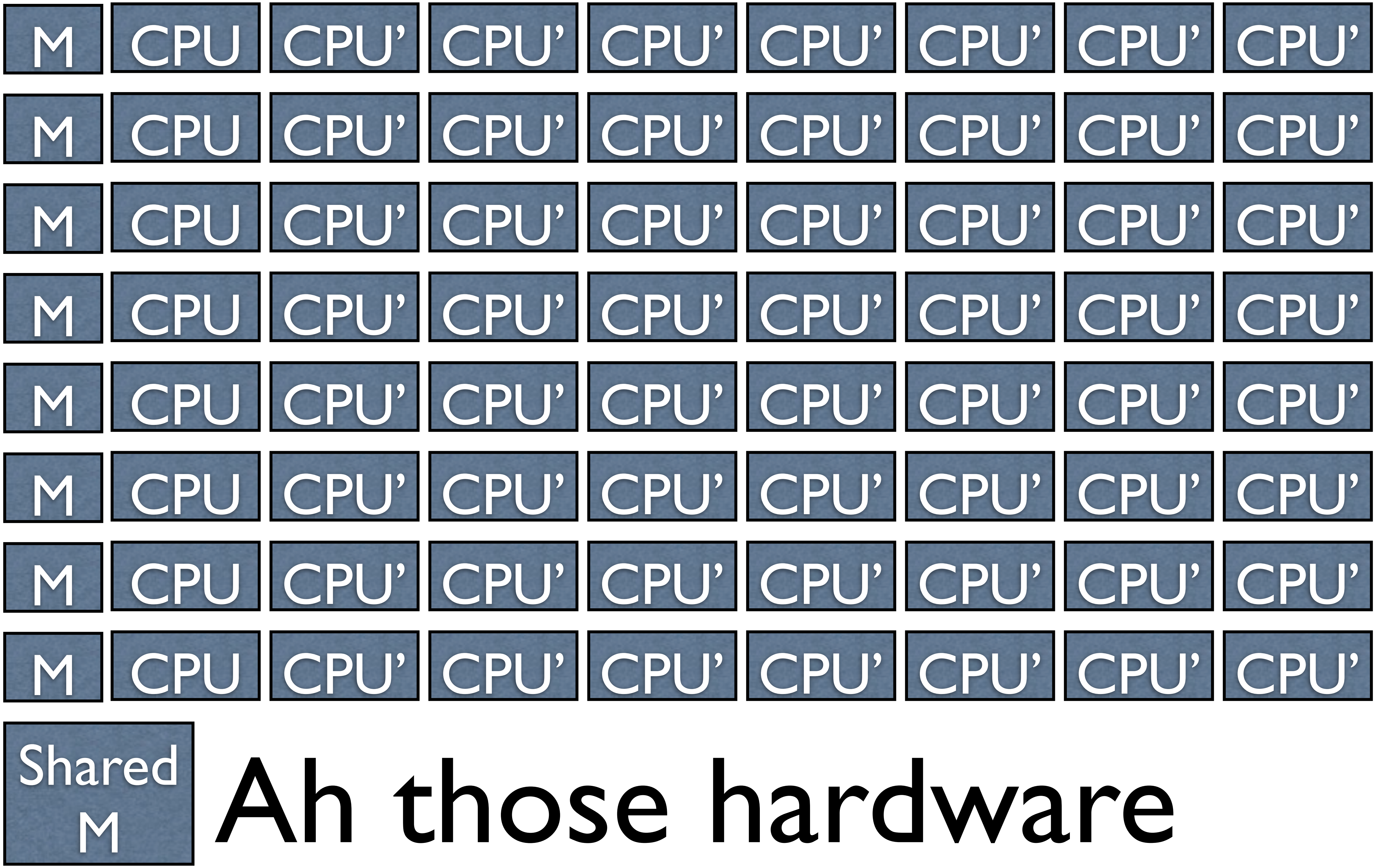
The sad reality



multicore ... can be the same



Or worse (e.g. cell)



M CPU CPU’ CPU’ CPU’ CPU’ CPU’ CPU’ CPU’

M CPU CPU’ CPU’ CPU’ CPU’ CPU’ CPU’ CPU’

M CPU CPU’ CPU’ CPU’ CPU’ CPU’ CPU’ CPU’

M CPU CPU’ CPU’ CPU’ CPU’ CPU’ CPU’ CPU’

M CPU CPU’ CPU’ CPU’ CPU’ CPU’ CPU’ CPU’

M CPU CPU’ CPU’ CPU’ CPU’ CPU’ CPU’ CPU’

M CPU CPU’ CPU’ CPU’ CPU’ CPU’ CPU’ CPU’

M CPU CPU’ CPU’ CPU’ CPU’ CPU’ CPU’ CPU’

Shared M

Ah those hardware guys ...

Modern systems

- Multi-core, multi-socket
- Different ways of accessing memory (NUMA)
- Move towards heterogeneous processors
  - CellBe and Wirespeed
  - Save die size by having a few smart cores and lots of dumb ones
    - Dumb can be really, really dumb

Nix

Nix design space

- Multi-socket, multi-core
  - heterogeneous cores
- Not all memory is shared
- N^2 CPUS but N can run kernel code
  - Some might have only user-mode ISA
- First pass was on 24-core 4-socket K10

NIX Structure

- Cores have roles
- (Limited) Shared memory between cores
  - Used for comms
  - Confession: we don't always adhere to this rule
- Big pages
  - i.e. regular pages are 2MiB, big are GiB

Messaging: Inter-Core Call

- Essentially an active message
- Uses notion of small memory window between cores
- Trick: code address in all cores is same (works even if unshared memory)
    - So send address, not code, for active messages

```
struct ICC
{
        /* fn is kept in its own cache line */
        union{
                void    (*fn)(void);
                uchar _ln1_[ICCLNSZ];
        };
        int     flushtlb;       /* on the AC, before running fn */
        int     rc;             /* return code from AC to TC */
        char*   note;           /* to be posted in the TC after returning */
        uchar data[ICCLNSZ];    /* sent to the AC */
};
```

Core types

- TC == Timesharing Core
- AC == Application Core
- KC == Kernel Core

Timesharing Core

- Standard kernel
  - handles all hardware interrupts
  - schedules processes
  - driver interfaces

Application Core

- Runs user applications to completion/yield
  - Only FPU interrupts
  - New process state "Exotic"
- When idle, waits for ICC (message shown above)
  - polls ICC struct for non-nil fn pointer
- Currently: loaded from shared memory of TC
- Kernel context stays on TC

Kernel Core

- Running kernel tasks
  - kthreads that never context switch out
- a "noise" core
- No current use but we think there might be applications of it
- e.g. drivers without interrupts

One other rule

- Existing binaries run unchanged
- Instead of a "redo the world" approach
- Many such "redo the world" efforts, after a few years, have gone nowhere -- still no useful user mode, for example
- We wanted to avoid that situation

Implementation

Nix structure

- Inter Processor Interrupt to bring the cores up (standard approach, even in BIOS)
- infinite loop on ACs while they wait for work
  - i.e. ACs wait for ICC active message fn pointer to be non-NULL
- ACs Issue async system calls.

New calls

- execac(int core, char *path, char *args[]);
  - for non-shared-memory case
- rfork(RFCORE)
  - move from TC to AC
- rfork(RFCCORE)
  - move from AC to TC

System Calls

- System call code (kernel code) is not run on AC
- On K8, “Migrate” process from AC to TC, run call on TC, then move it back
  - significantly slower(~10x)
- We also have system call queues for K8
  - 2 Queues: request, reply; 2x slower
- On Cell-like systems, we would use Tubes (later) to forward system calls/connect to services

New “optimistic” semaphore

- void upsem(int *sem);
- void downsem(int *sem);
- int altsems(int *sems[], int nsems);
- Can proceed if no need to block
  - Else drop into kernel after several spins
- 124 lines user; 310 lines kernel
- Useful for supporting IPC such as ...

New IPC: Tubes

- Builds on sems
- Tube* newtube(ulong msz, ulong n);
- void freetube(Tube *t);
- int nbtrecv(Tube *t, void *p);
- int nbtsend(Tube *t, void *p);
- void trecv(Tube *t, void *p);
- void tsend(Tube *t, void *p);
- int talt(Talt a[], int na);

Uses of cores

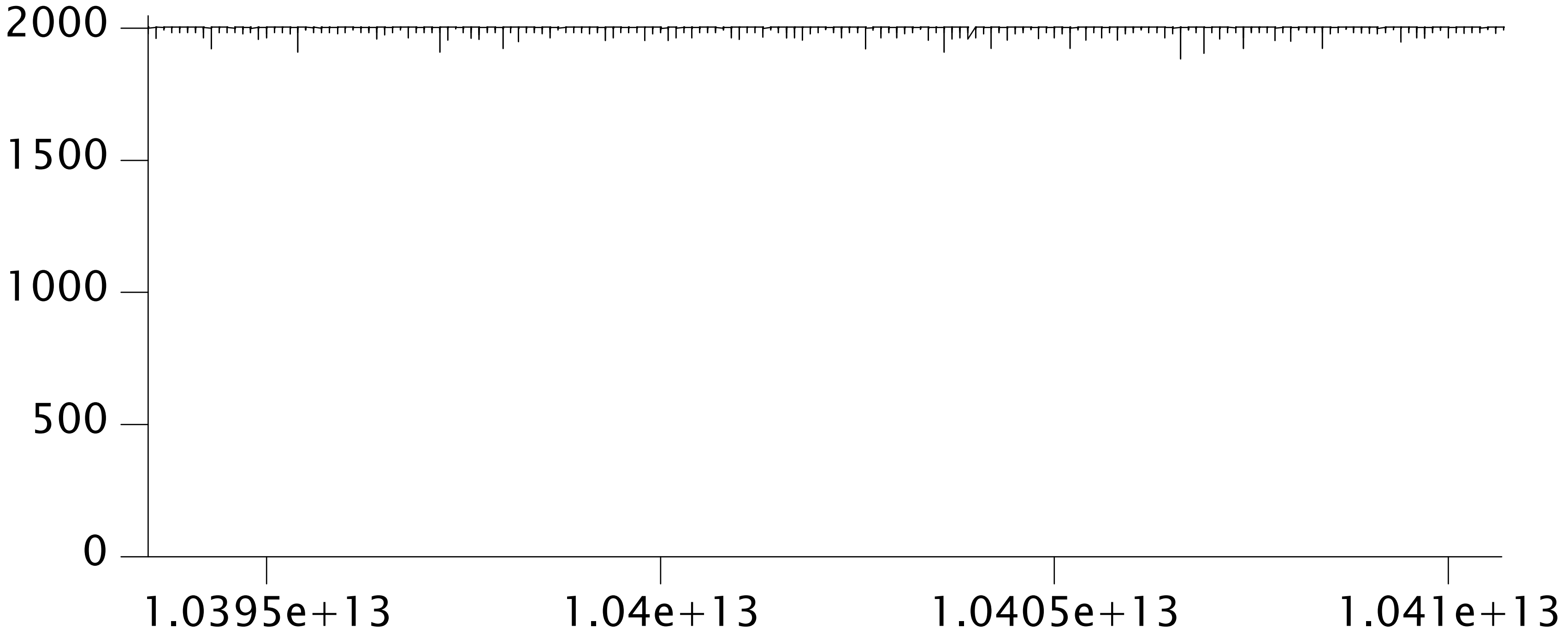
- Explicit request to run on an AC
- Implicit: consume enough full quantas, get moved to AC.
- Issue enough system calls, get moved to TC.
- Note the rule: use *more* CPU, *get even more* CPU: exact opposite of today

Evaluation

Fixed Time Quantum: FTQ

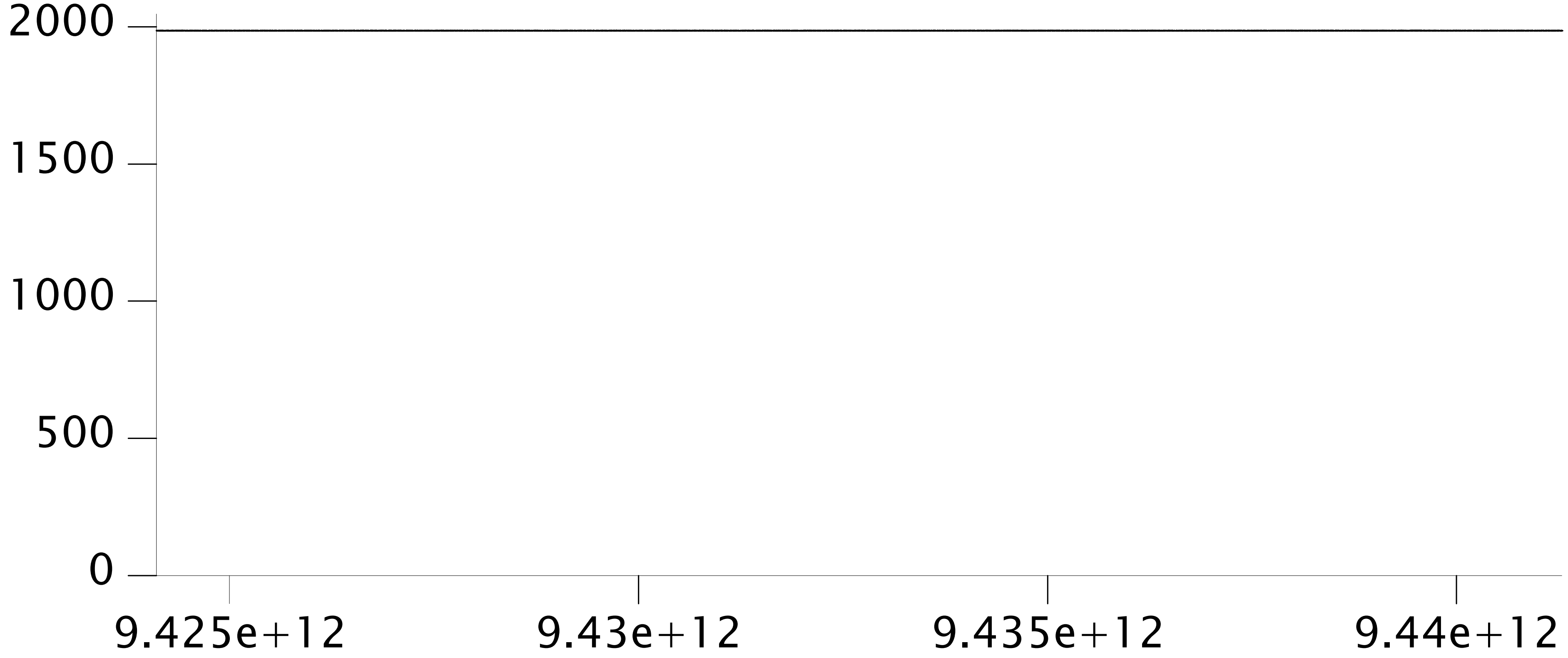
- Measure work per quanta (work/deltaT)
  - Careful implementation to ensure stationary, statistically useful time-series data
  - We've used it to find interference no on suspected (e.g. on Purple)
- Quantifies OS noise
- Used by Cray, IBM, DOE Labs, others

TC(Plan9)



FTQ on a TC
Work vs. Cycles
Smoother is better

AC



FTQ on an AC
Work vs. Cycles
Smoother is better

Current state

- Kernel and ACs work.
- 64 bit address space works (a bit better than on Linux)
- Transparent GiB PTEs went in 9/1/1
  - The AMD64 is a train wreck, gcc more so
  - But the Nix AMD64 support is far better
- In-kernel Linux emulation almost done

No kludgery needed (unlike Linux ...)

```
/* get top of heap */
p = segbrk(0, 0);
va = (uintptr)p;
/* round up to 64 GiB  (G as in Billion) */
va = ROUNDUP((va), 1ULL*GiB) + 64*GiB;

/* morecore */
np=(void*)va;
p = segbrk(p, np);
```

TODO

- AC statistics in /proc
- KC usage (implementation done)
- note (signal) handling on ACs
- optimization
- Power management

Conclusion

- New OS for future manycore CPUs
  - Differentiated cores
    - TC, AC and KC
  - Handle memory sizes
- Keep things working: Current implementation much faster for CPU bound tasks while still backwards compatible
  - If an AC is not available process works fine on a TC
- 400 new lines of assembly; 40 lines of “port”; 350 lines of AMD64 support (huge pages etc.)
- Far more work spent in design than actual coding