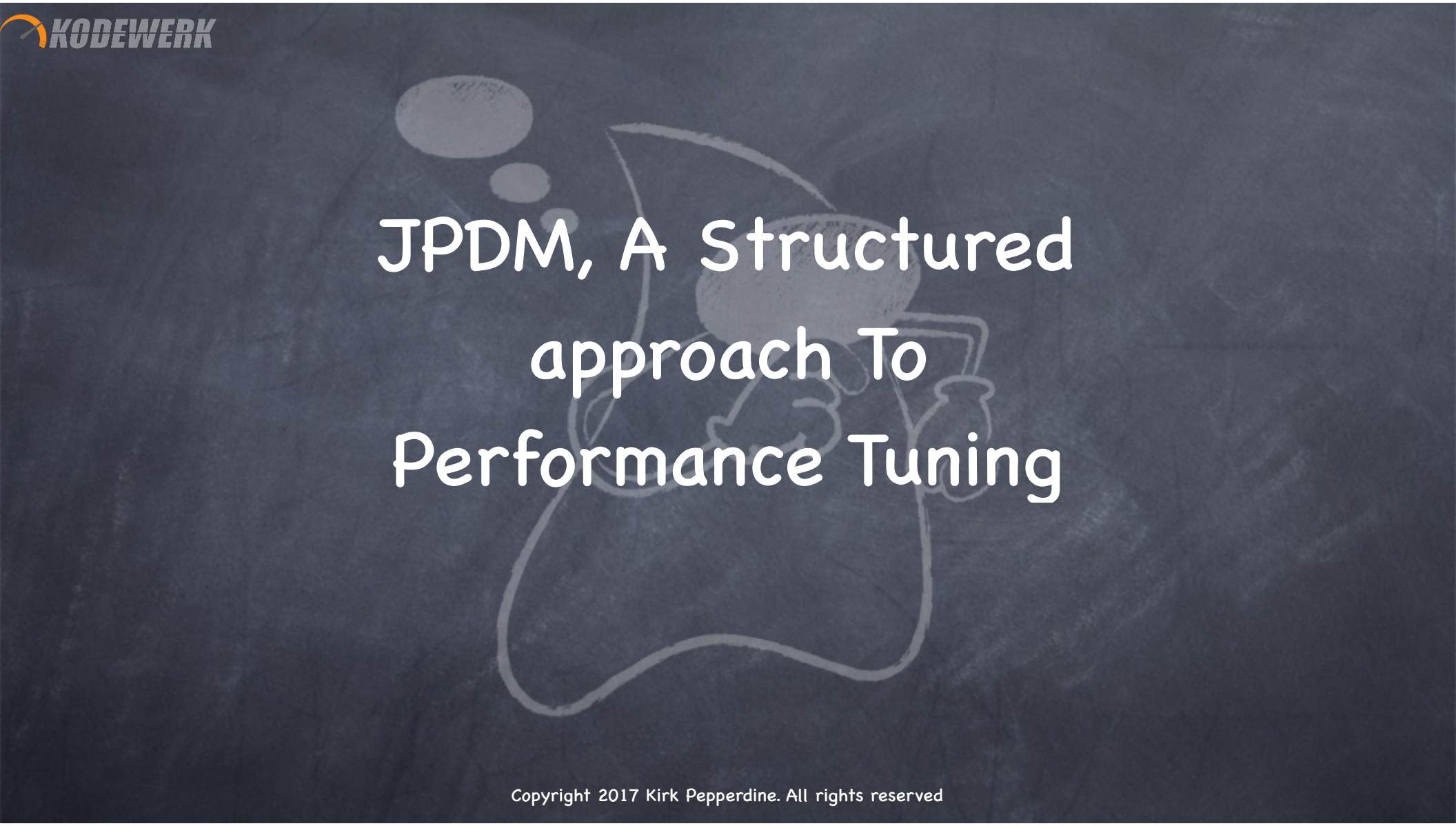
KODEWERK

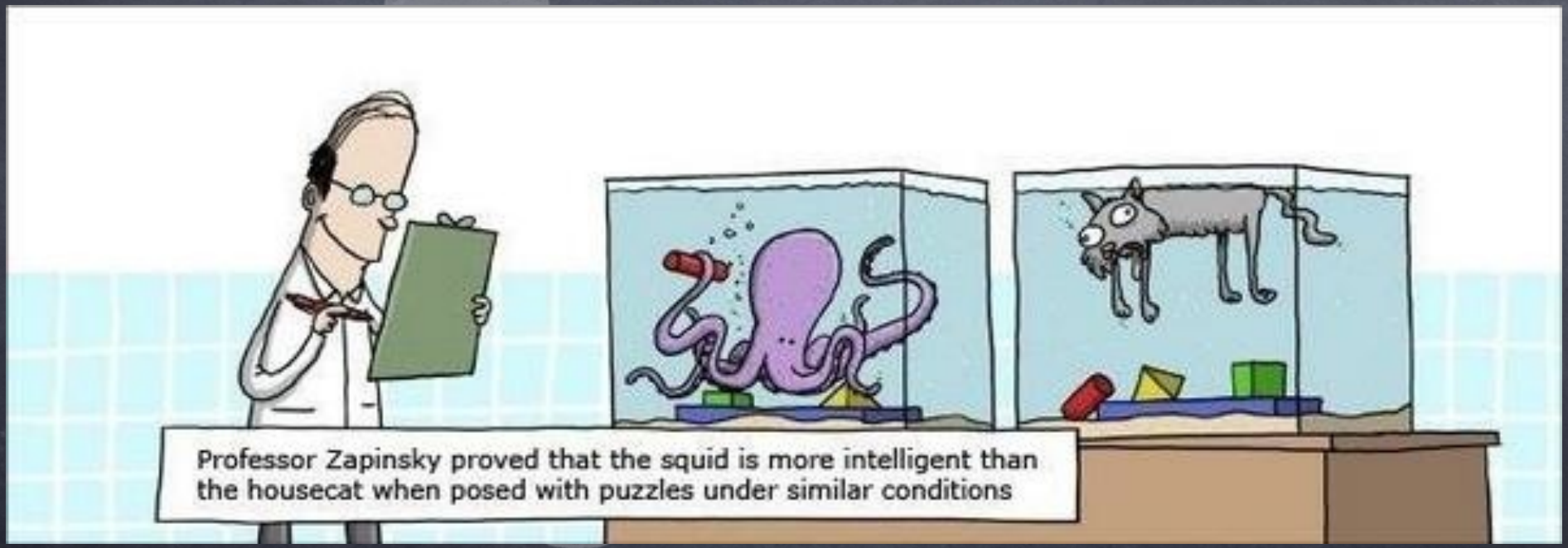


JPDM, A Structured approach To Performance Tuning


About Me
Kodewerk Ltd Performance Consulting
Java Performance Tuning Workshops
Co-Founded jClarity
Java
Champions

KODEWERK

Disclaimer



KODEWERK

Our Typical Customer

- Application isn't performing to project sponsors expectations
  - users wait for minutes for the application to respond

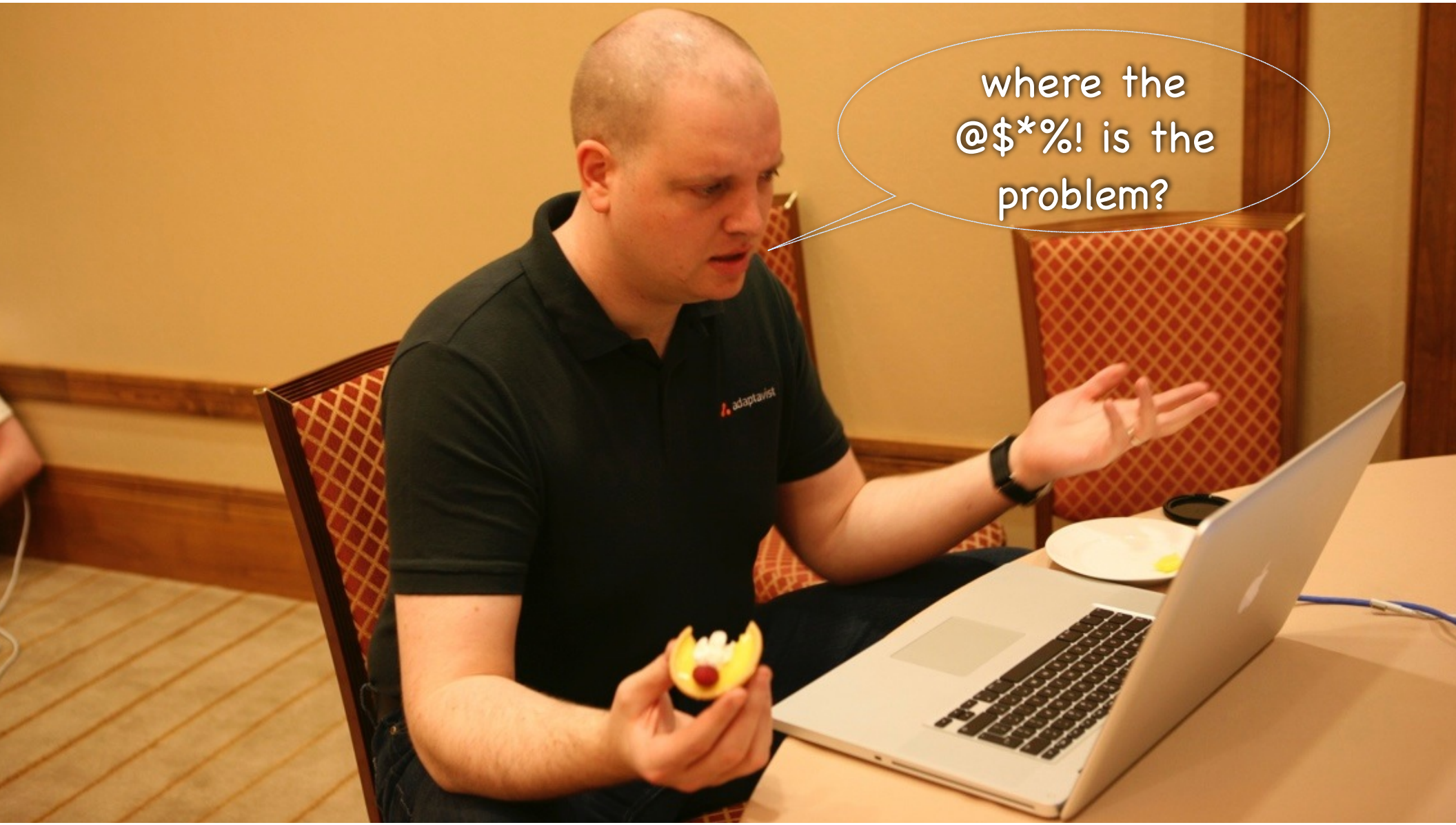

iPad

KODEWERK

Developers Get Involved

StringBuffer is being used all over the place...
We need to change it to StringBuilder

Looks like a database problem, we need to migrate to [buzzname goes here]


where the @$*%! is the problem?

KODEWERK

Managers Get Involved

Form a tiger team

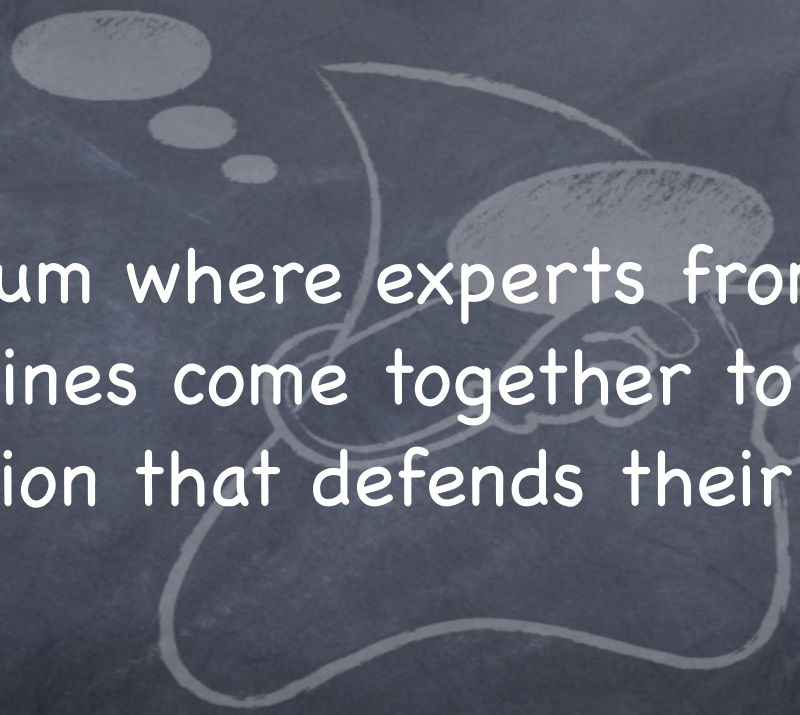
KODEWERK

What is a Tiger Team?

A team of specialists in a particular field brought together to work on specific tasks.

KODEWERK

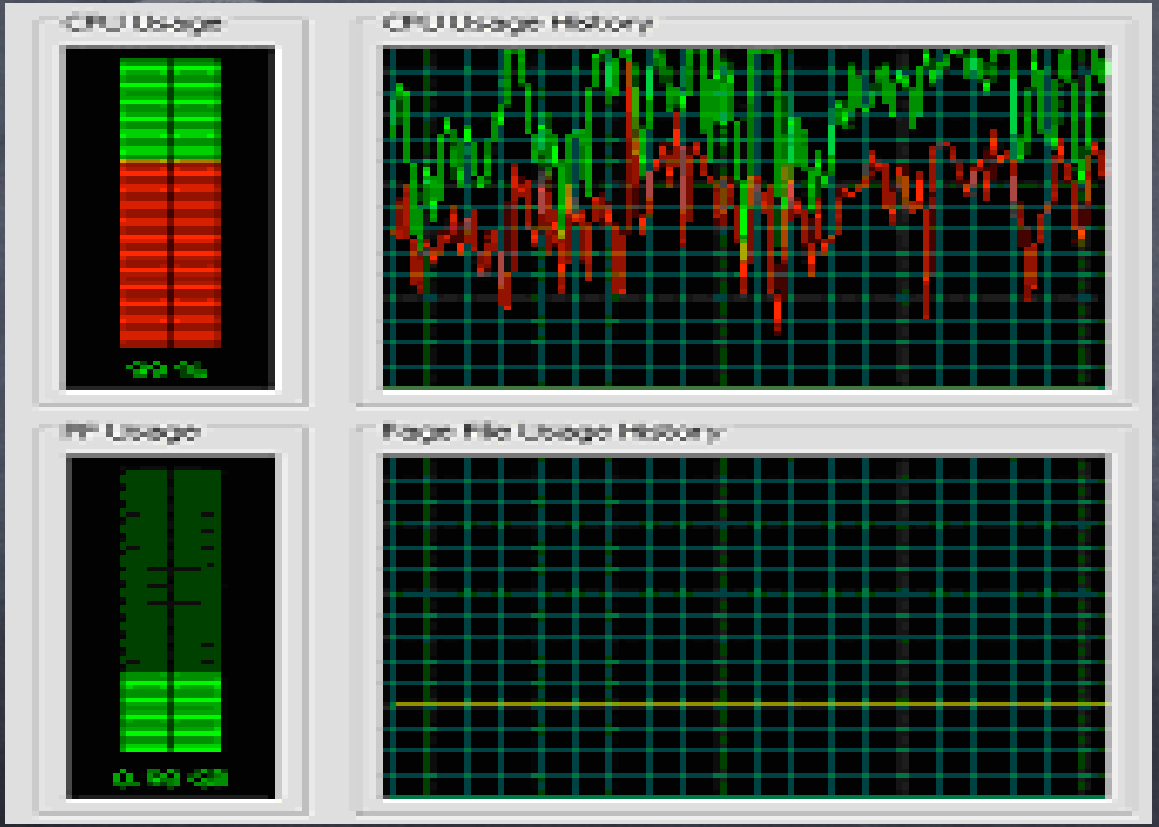
What is a Tiger Team?



A forum where experts from different disciplines come together to express an opinion that defends their specialty

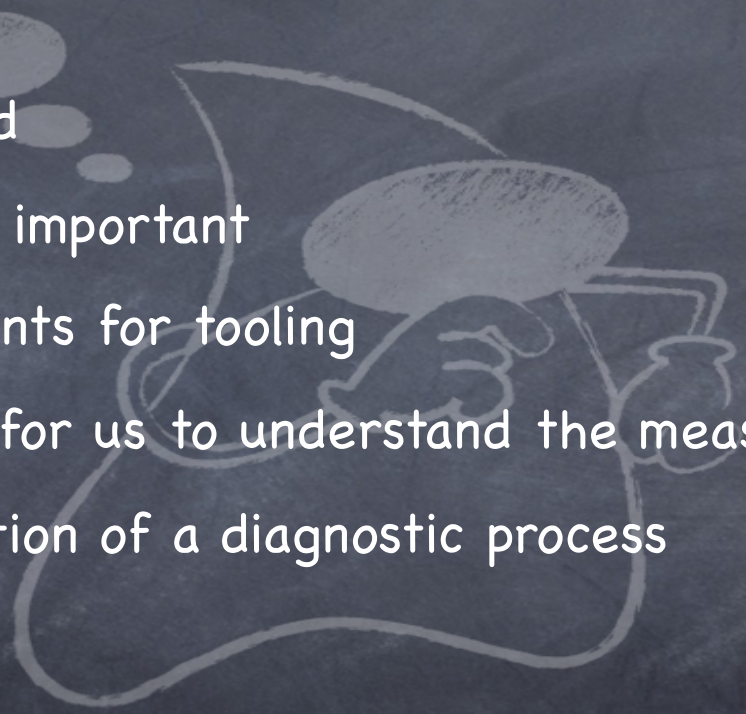
KODEWERK

Measure Don't Guess®



KODEWERK

Why a System Model?



- Help us to understand
  - what measures are important
    - defines requirements for tooling
  - provides a context for us to understand the measures
  - facilitate the definition of a diagnostic process

- Java Performance Diagnostic Model (JPDM)

KODEWERK

Where do Developers Live?

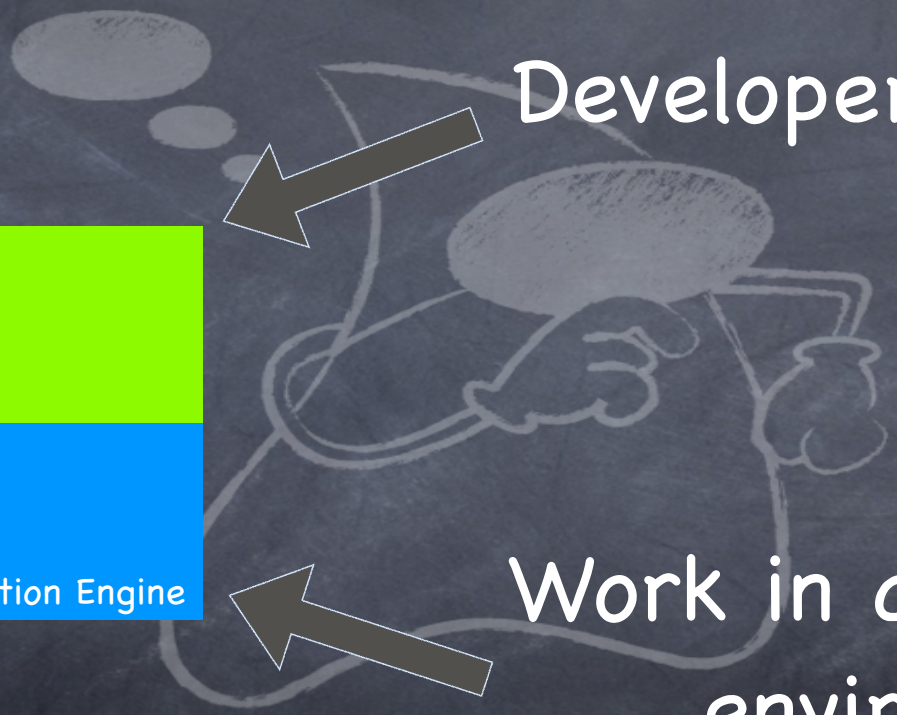
KODEWERK

Developers Live Here

Application

Code (algorithms)

KODEWERK



Developers Live Here

Application

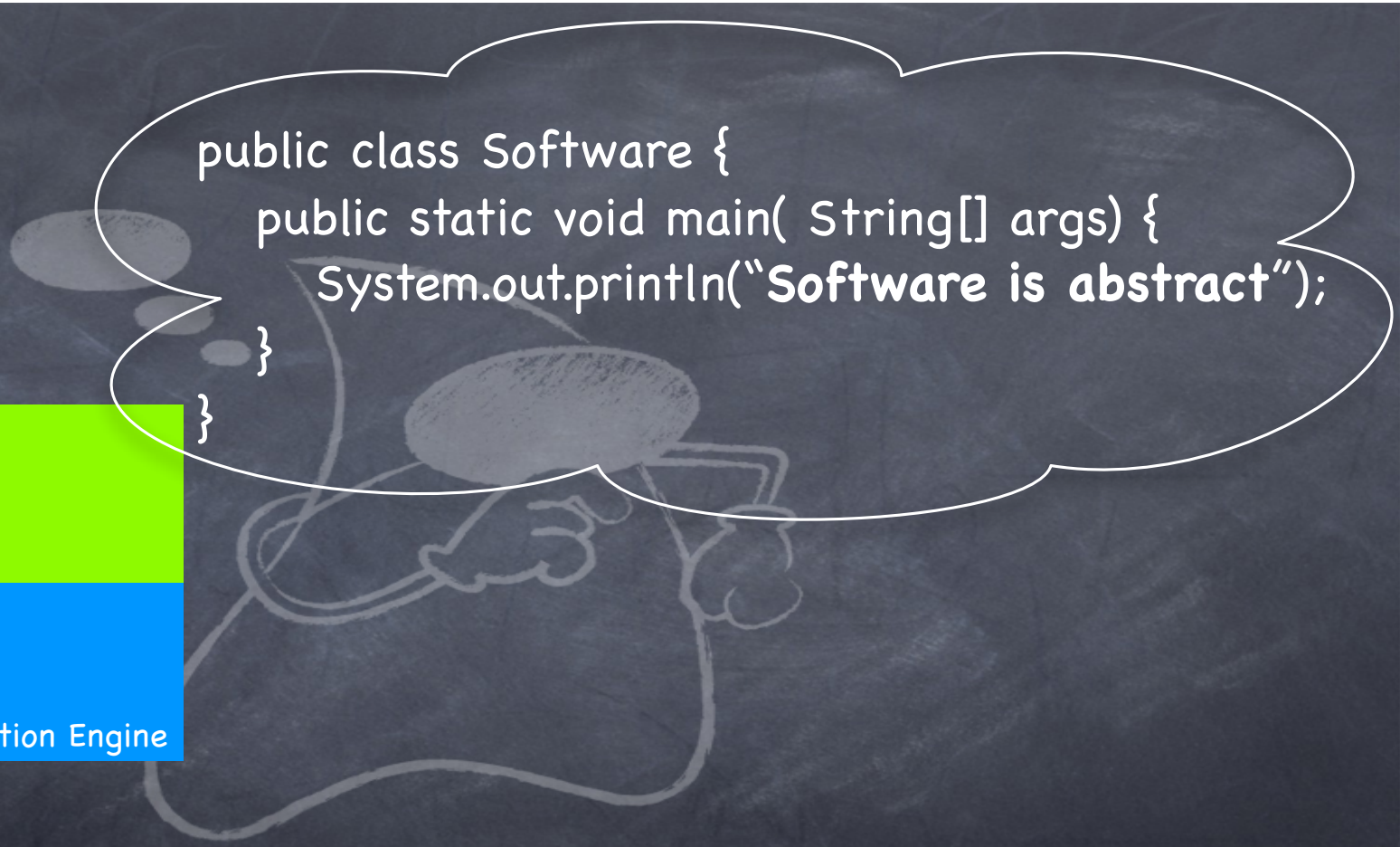
Code (algorithms)

JVM

Managed Memory, Execution Engine

Work in a virtualized environment

KODEWERK



Application

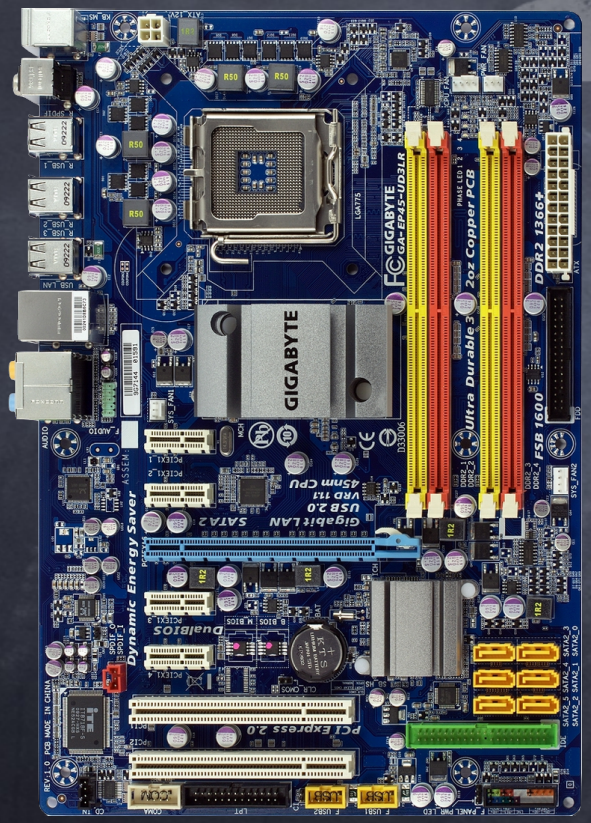
Code (algorithms)

JVM

Managed Memory, Execution Engine

KODEWERK

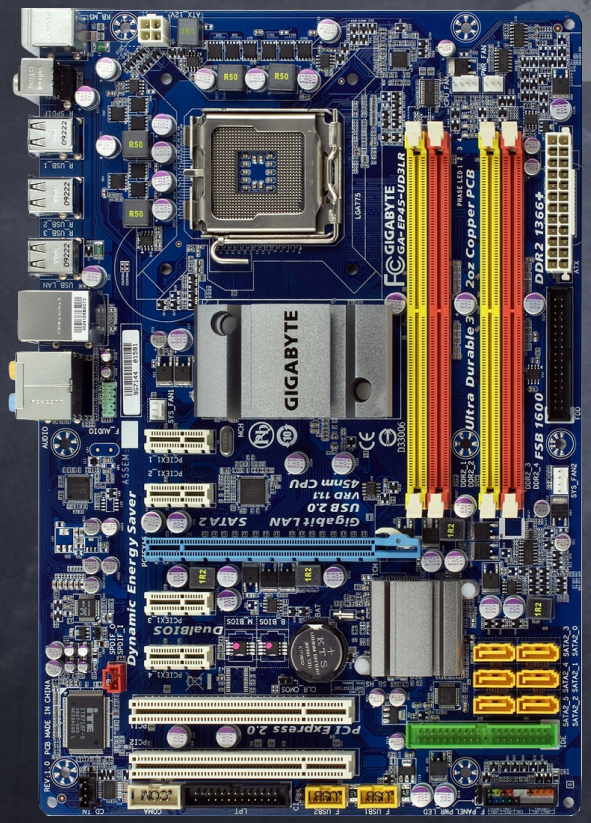
Hardware is Real!



- capacities
- volume
- throughput
- clock speed
- granularity
- cache line, sector size

KODEWERK

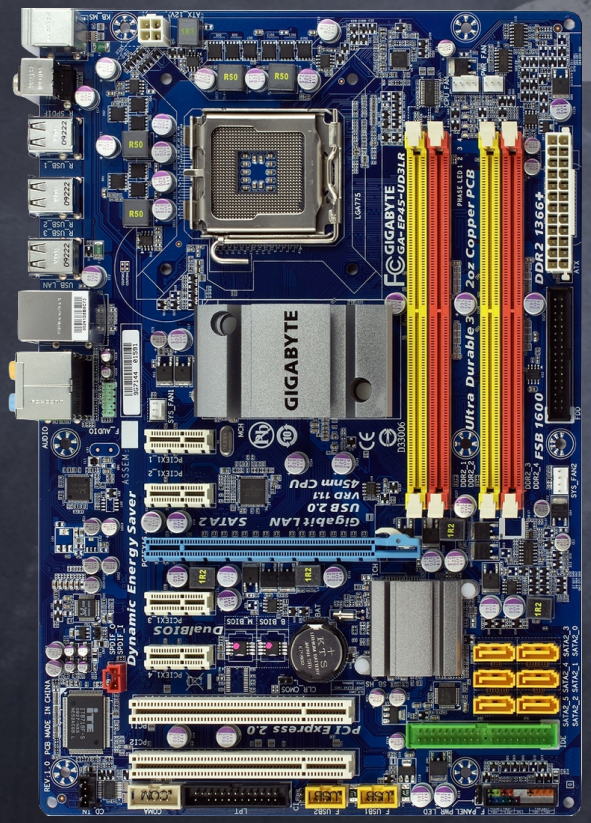
CPU



- capacity : number of cores
  - number of units in ALU
  - size of caches
- throughput : clock speed, CPI
  - bandwidth on various pipes (QPI)
- granularity : cache line size

KODEWERK

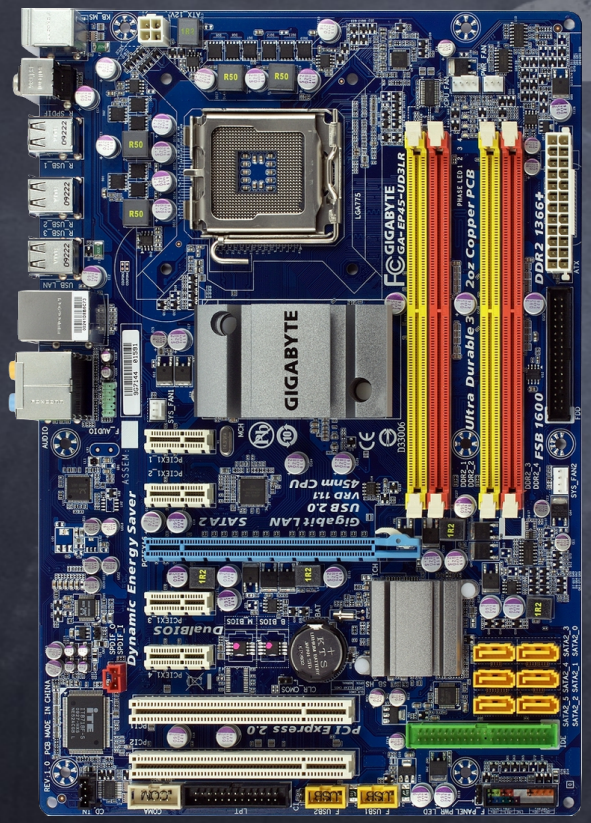
Memory



- capacity : number of bytes of RAM
- throughput : clock speed of BUS
  - tempered by bank cool of time
- chunk size : cache line size
  - typically 8 reads per cache line

KODEWERK

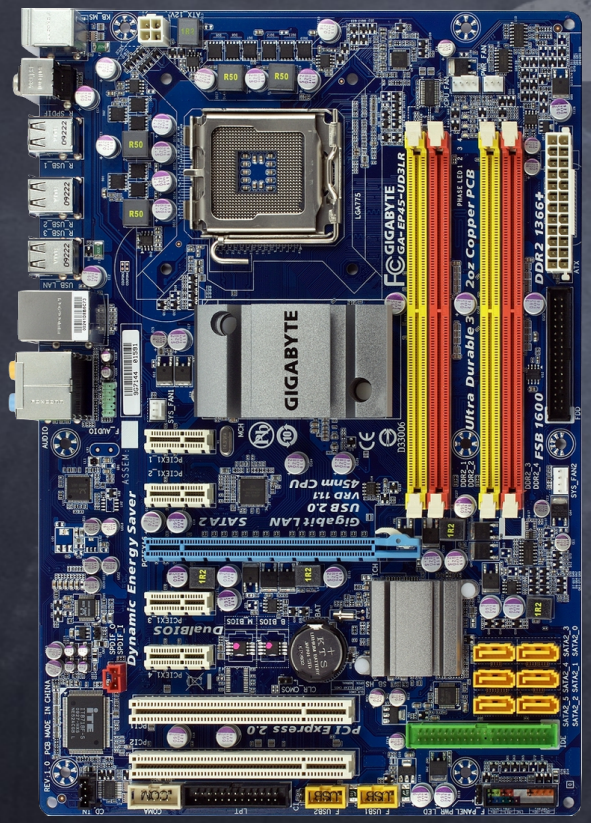
Disk



- capacity : number of bytes of RAM
- throughput : controller clock speed
  - ~1 Bbit/sec (SATA ~3Gbits/sec)
- granularity : disk sector (512 bytes)
- other latencies
  - sweep arm speed

KODEWERK

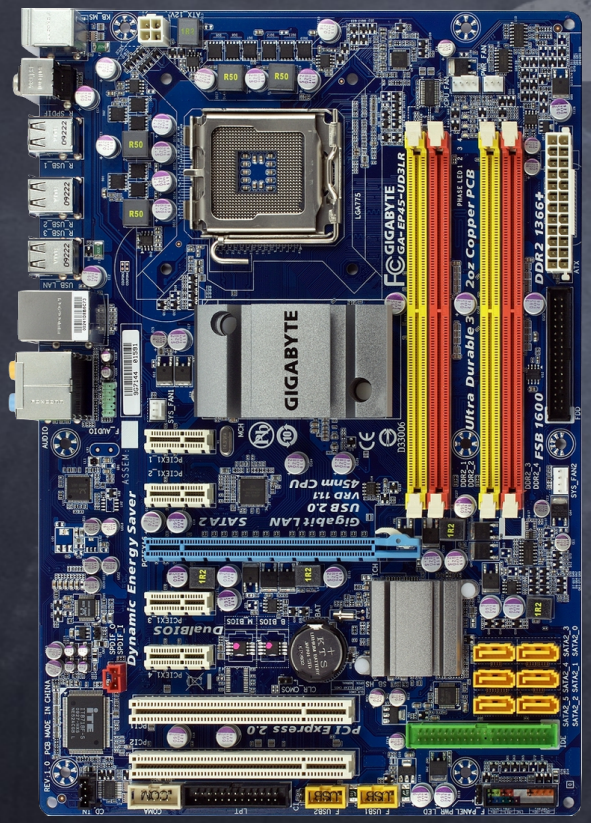
Network



- capacity : 1 per network card
- bandwidth : maximum volume of data transferred per second (10^9 bits/sec)
- throughput : depends on protocol overheads
- granularity : depends on protocol
  - tcp payload is typically 1500 bytes

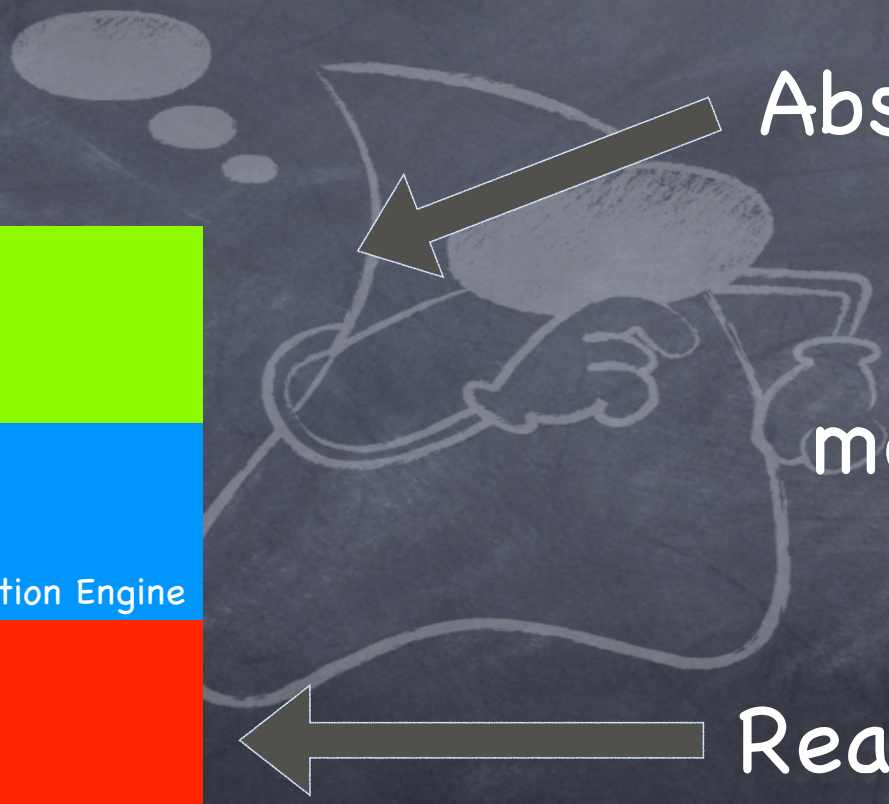
KODEWERK

Other Limits



- Other hardware devices
  - eg, video, sound cards
  - GPU
- Heat
- Battery capacities

KODEWERK



Abstract

meets

Reality

Application

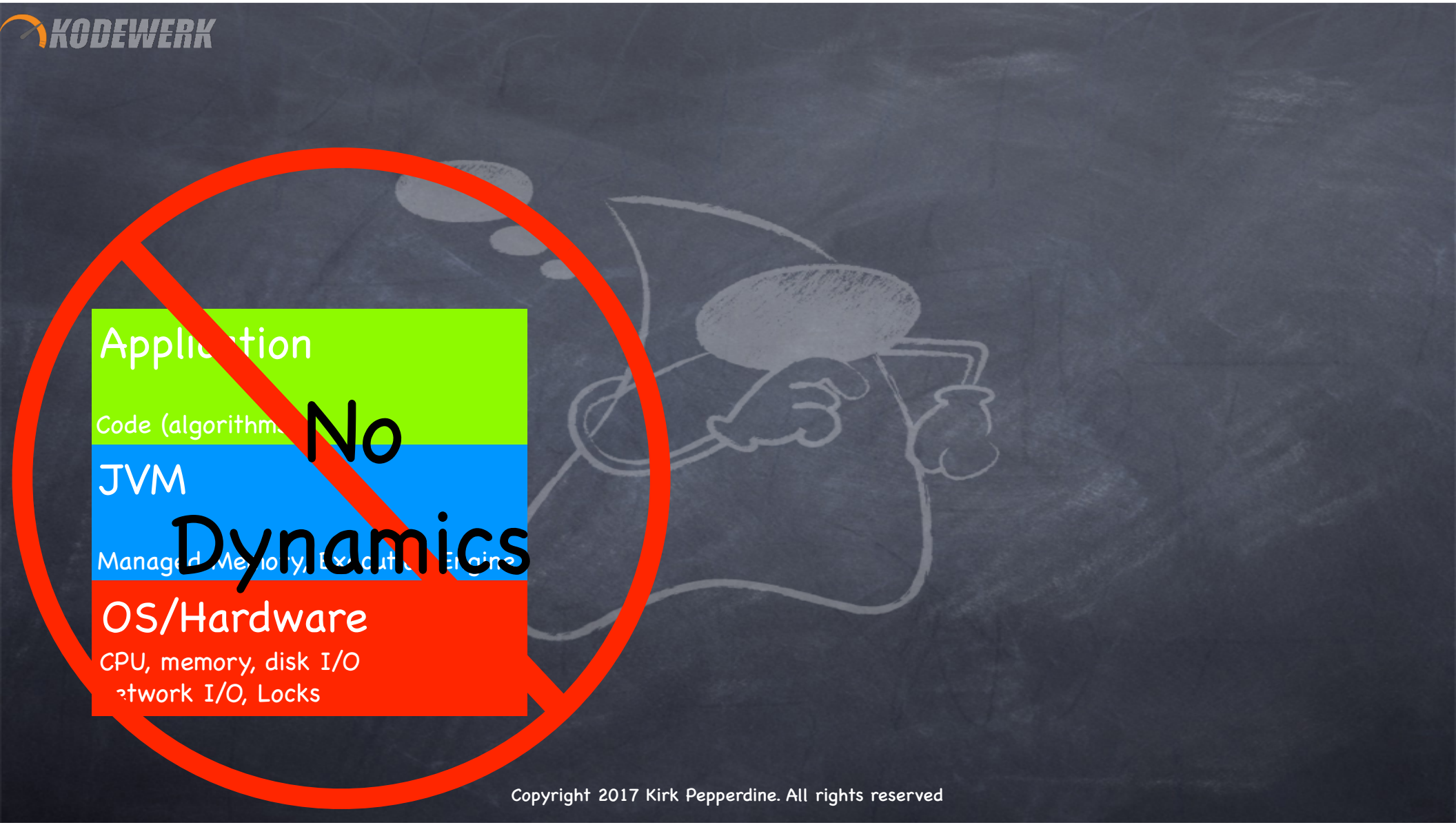
Code (algorithms)

JVM

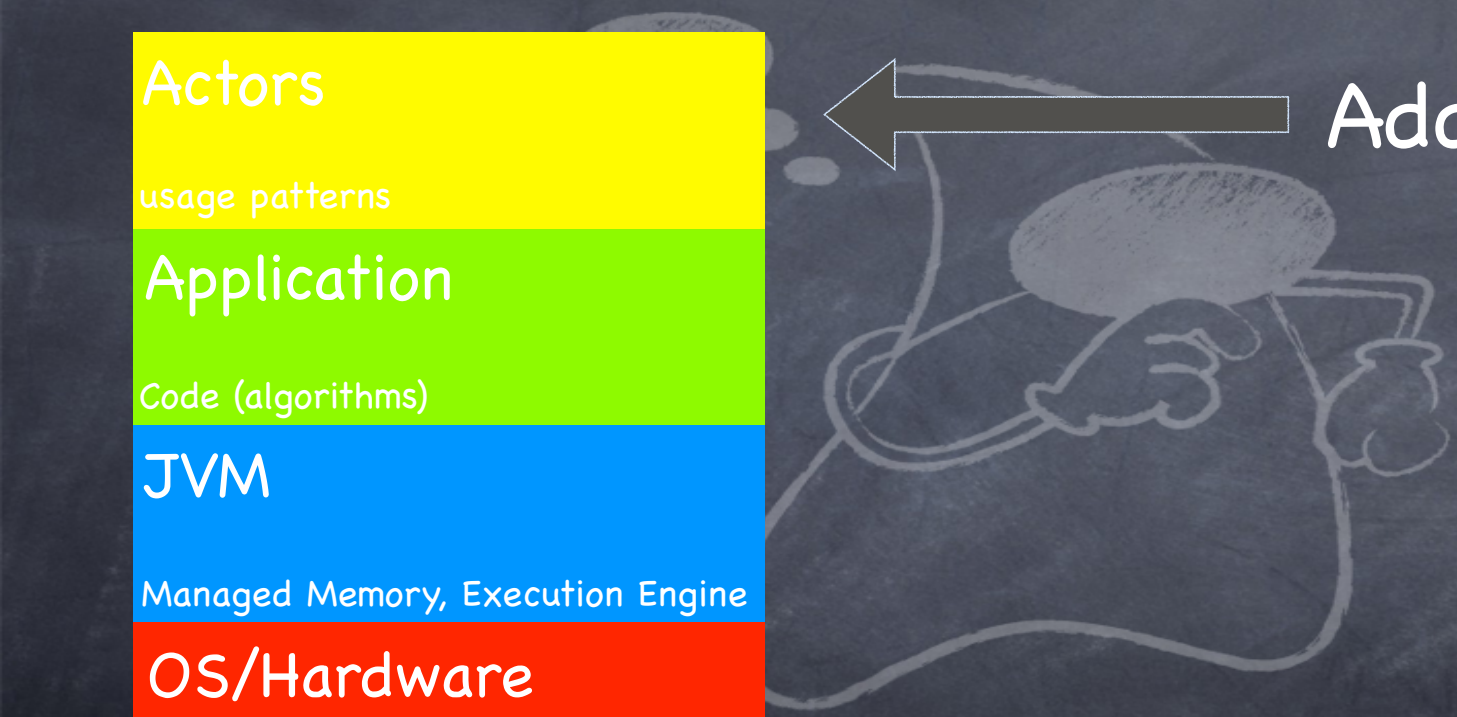
Managed Memory, Execution Engine

OS/Hardware

CPU, memory, disk I/O
network I/O, Locks



KODEWERK



Actors

usage patterns

Application

Code (algorithms)

JVM

Managed Memory, Execution Engine

OS/Hardware

CPU, memory, disk I/O
network I/O, Locks

Add dynamics

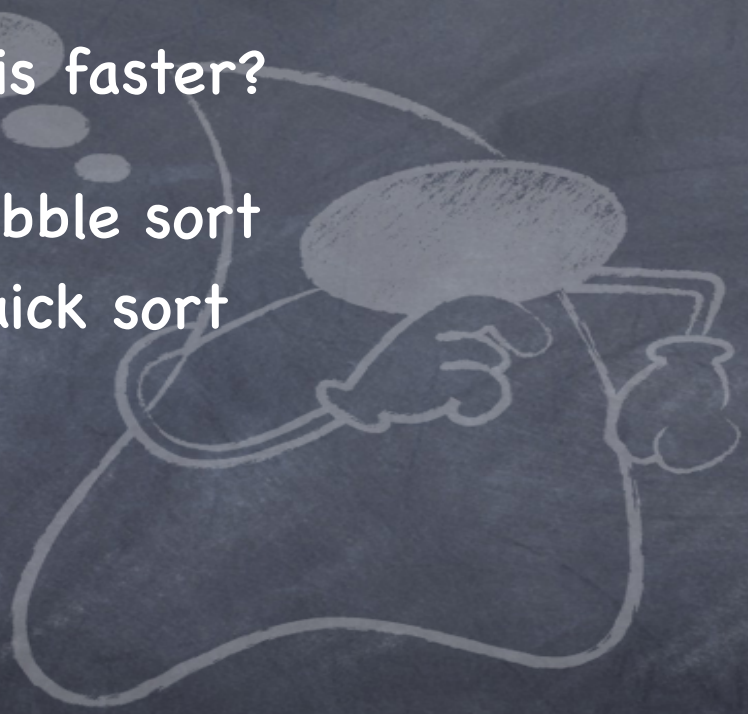
KODEWERK

Question?

Which is faster?

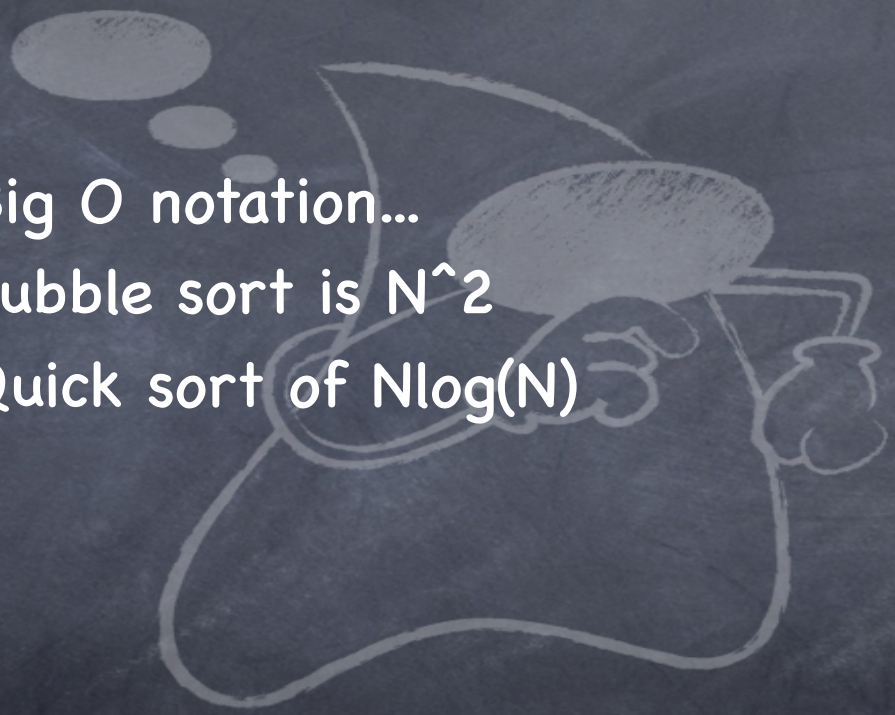
a) Bubble sort

b) Quick sort



KODEWERK

Hint

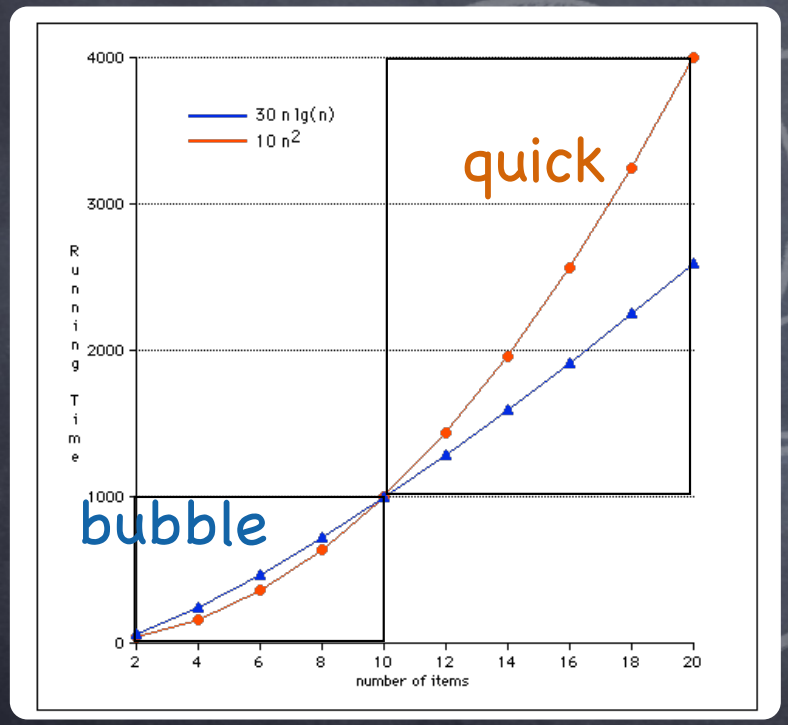


In Big O notation...

- Bubble sort is N^2
- Quick sort of Nlog(N)

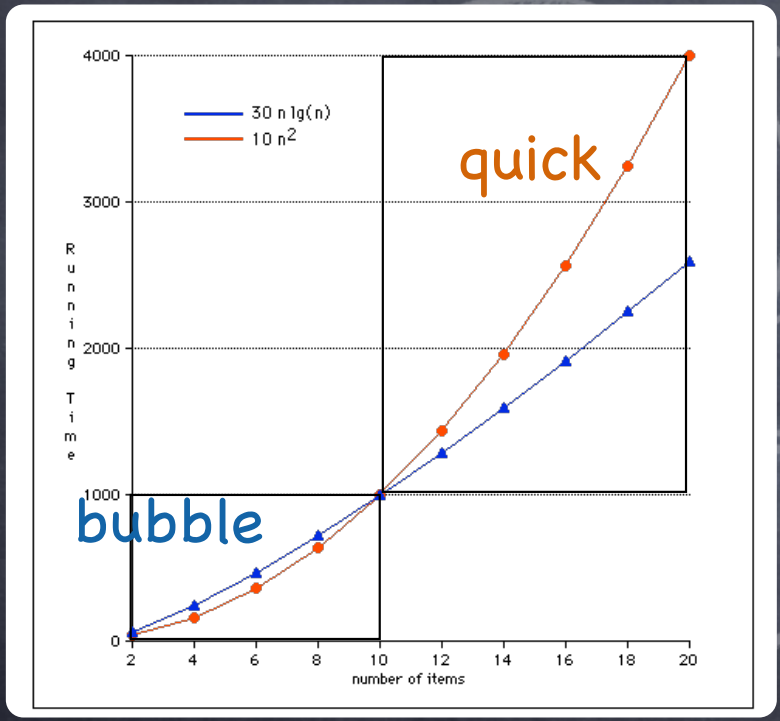
KODEWERK

However



KODEWERK

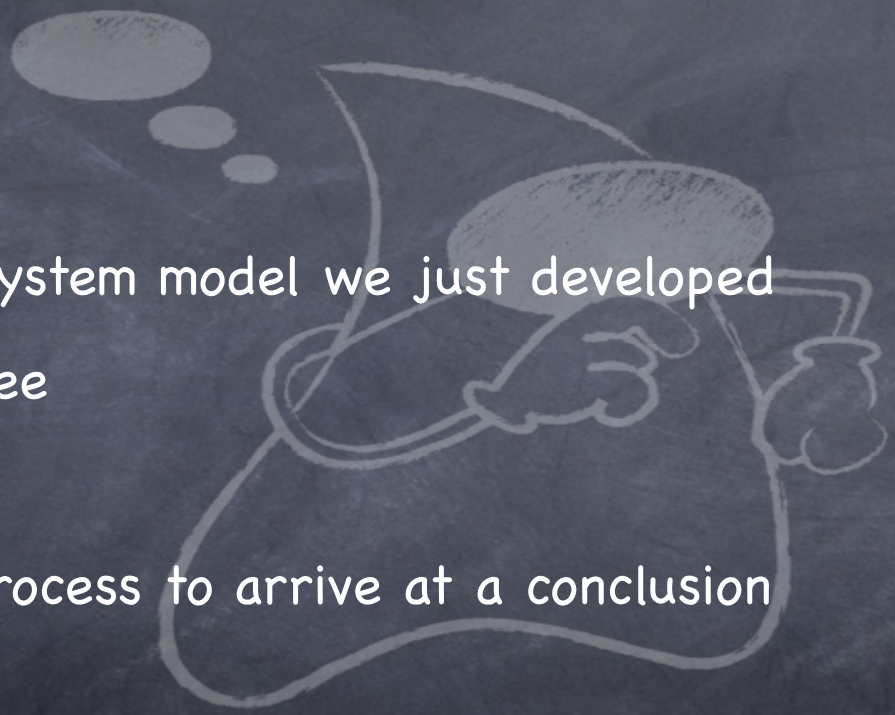
However



Number of items comes from the actors

KODEWERK

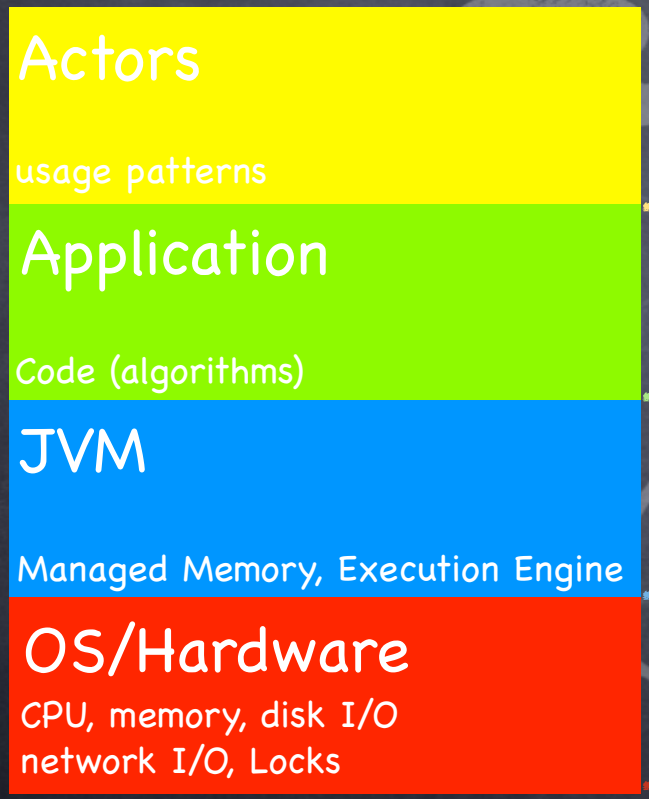
Performance Tuning Methodology



- Based on the System model we just developed
  - hypothesis free
  - methodical
    - step wise process to arrive at a conclusion

KODEWERK

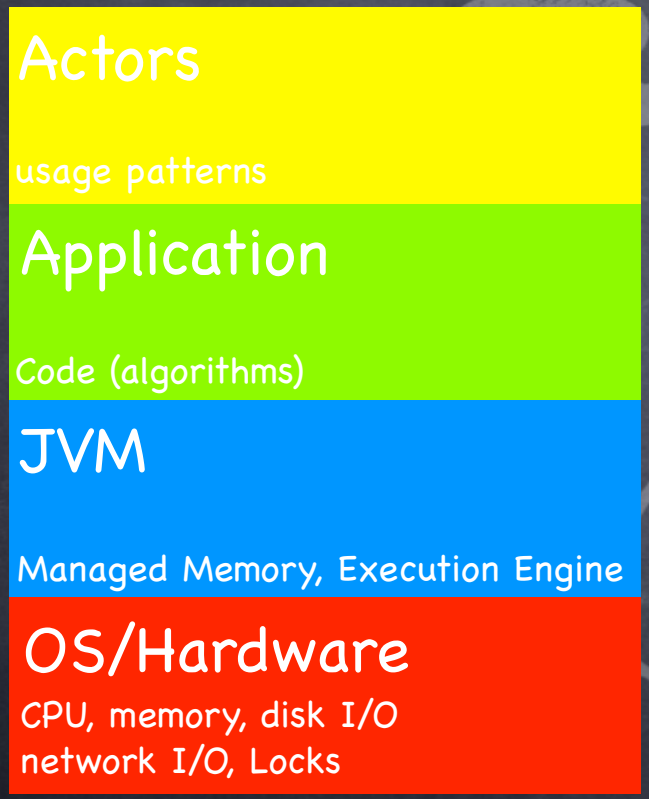
Hardware Consumption



- Actors drive the application
- Application drives the JVM
  - function of how actors interact with application
- JVM assisted by OS consumes Hardware
  - function of how application is coded
- Hardware is consumed
  - consumption limited by capacity
  - pattern of consumption is important

KODEWERK

Dominating Consumer

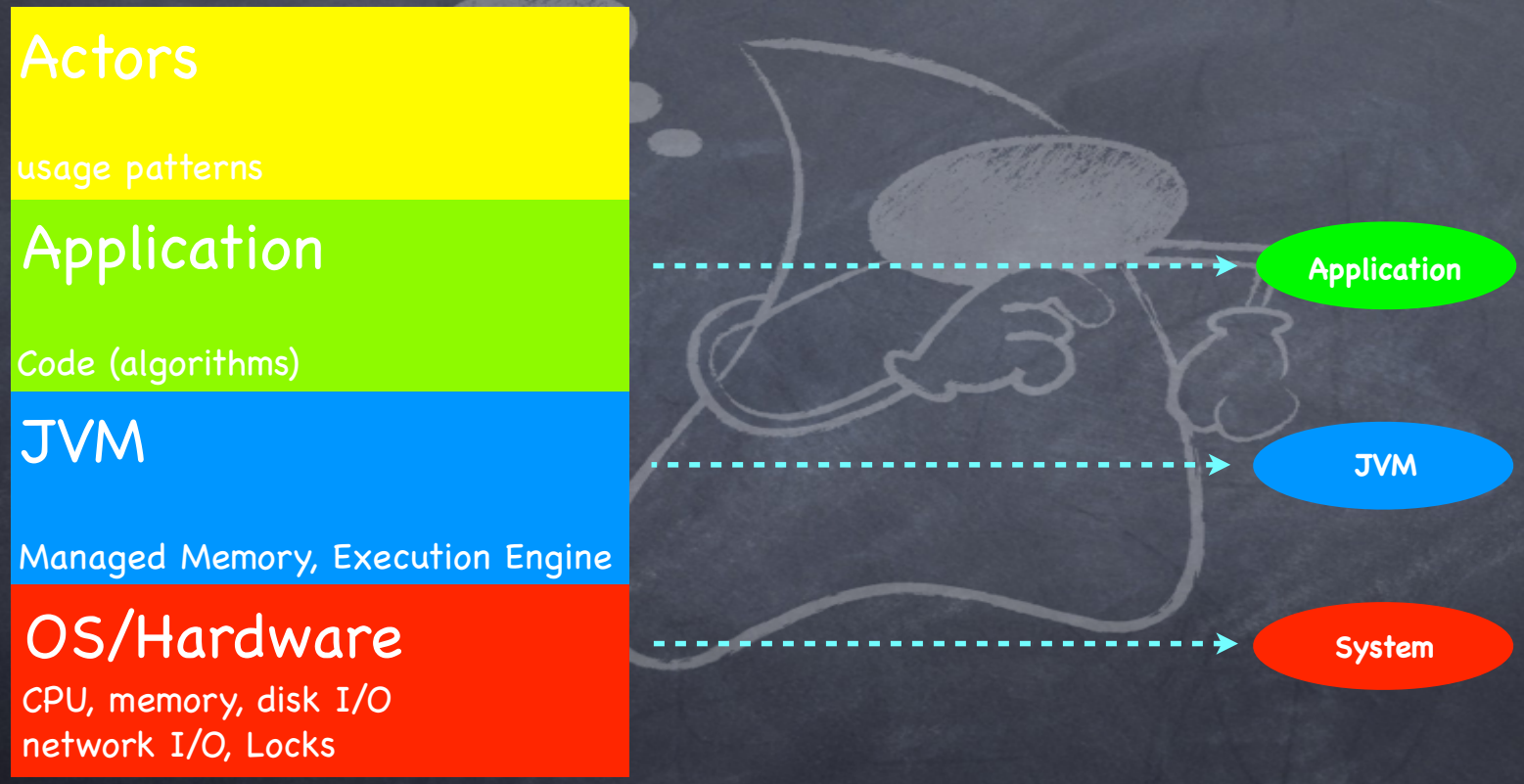


- Activity that dominates how the CPU is utilized
- Determination dominator by analyzing
  - CPU counters
  - Garbage collection logs

garbage collection logs

KODEWERK

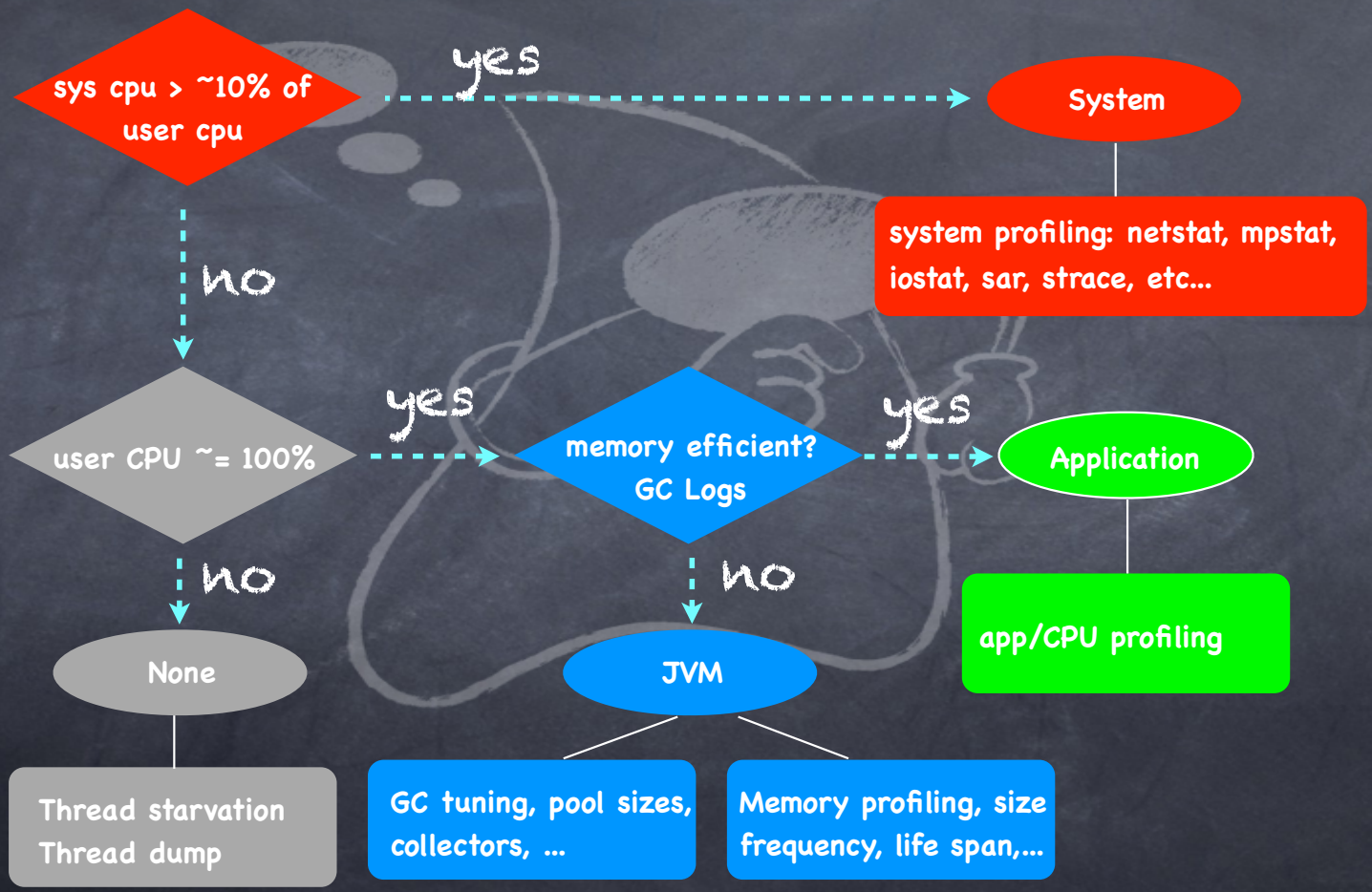
Dominating Consumers



• garbage collection logs

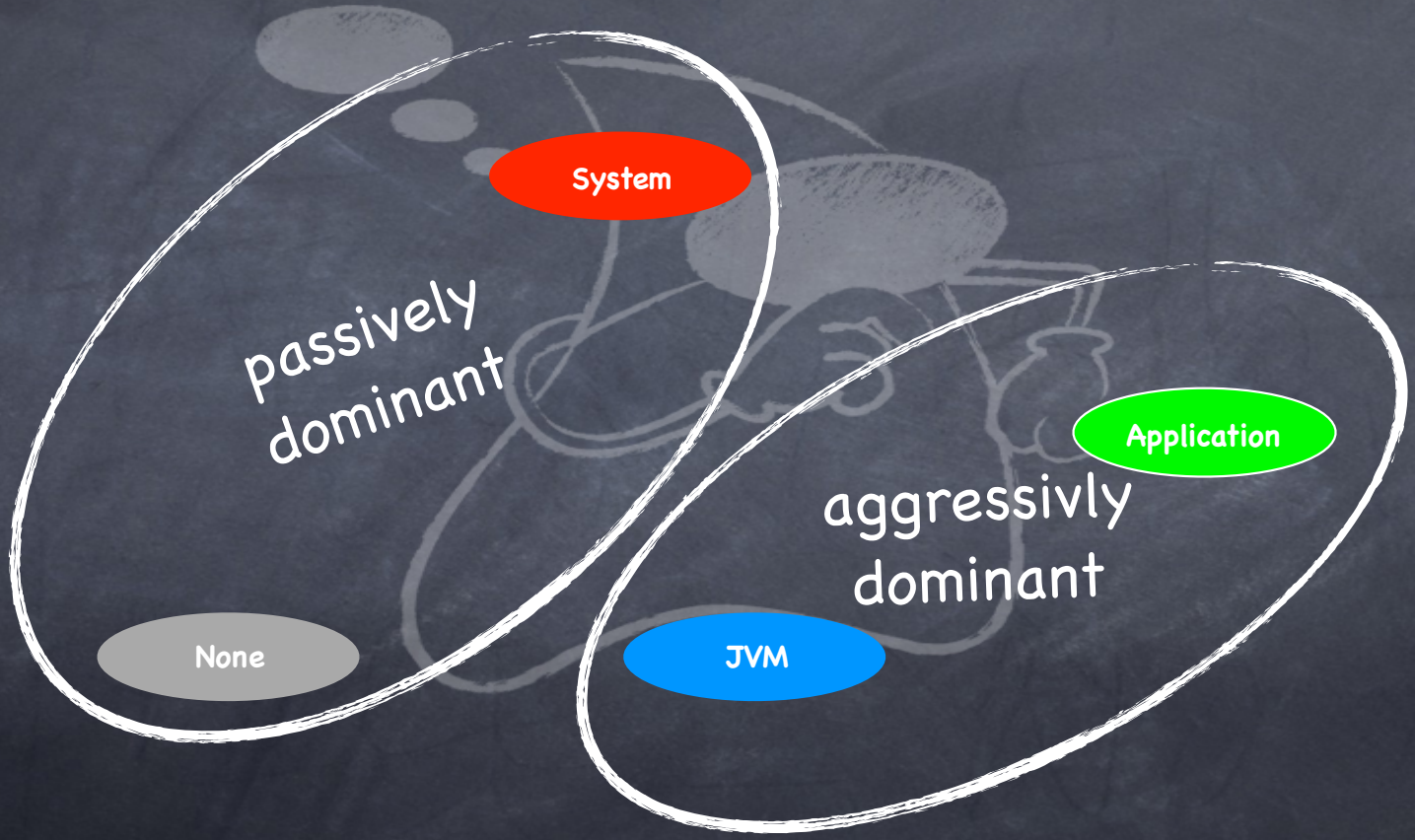
KODEWERK

Analysis



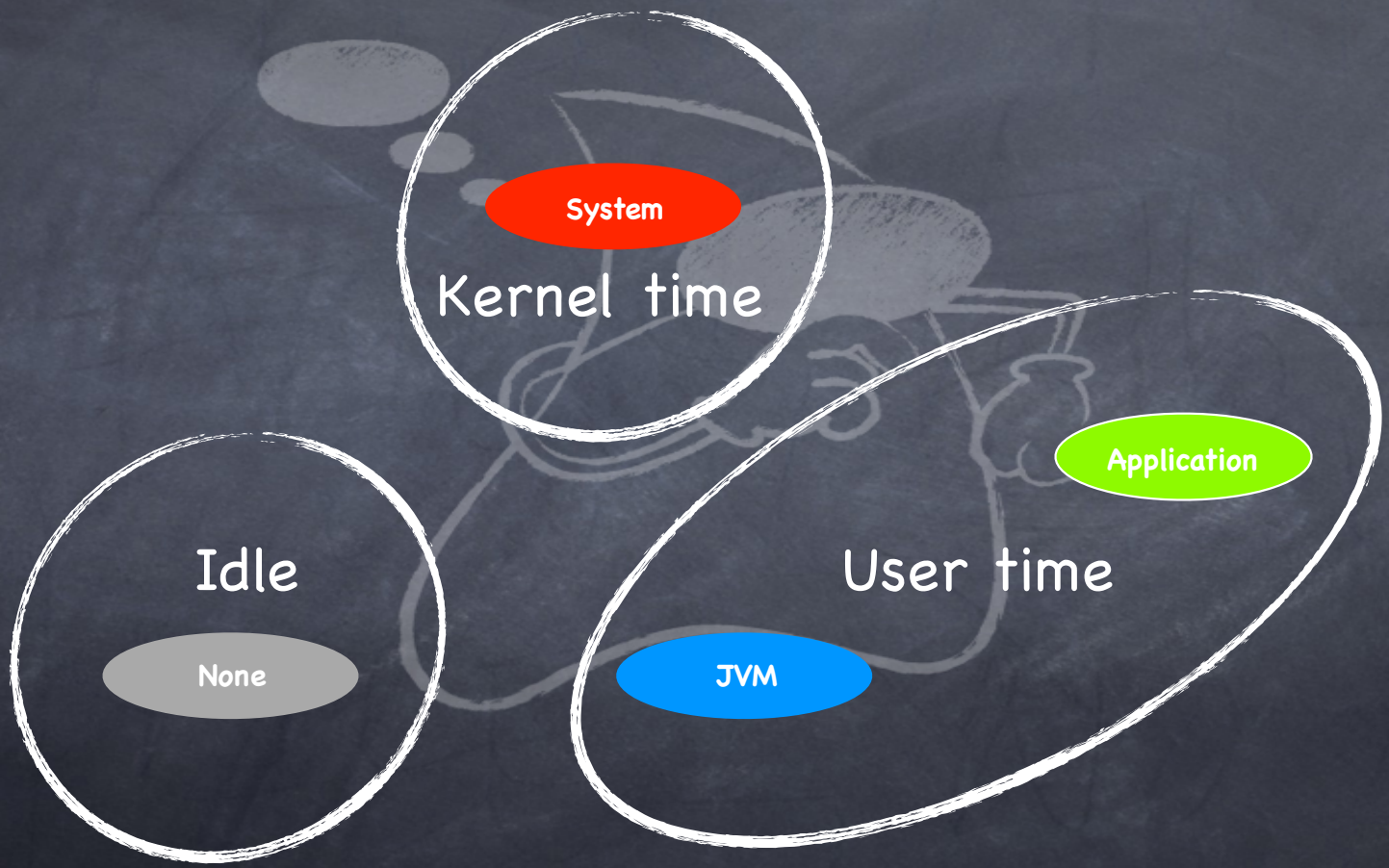
KODEWERK

Expression of CPU Consumption



KODEWERK

Measuring Consumption



KODEWERK

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa
3	9	100	24496	11096	13267036	0	0	0	5	0	1	2	1	96	1
3	2	100	23420	11088	13268328	0	0	0	0	77330	175352	17	26	39	17
3	9	100	20836	11088	13270628	0	0	0	68	105118	227382	14	40	21	25
8	4	100	23356	11080	13268272	0	0	0	0	80062	164387	12	30	29	30
7	7	100	23180	11084	13267068	0	0	0	72	98353	234851	15	43	28	15
11	2	100	25820	11088	13263676	0	0	0	120	100749	214921	11	42	17	30
13	1	100	22316	11088	13267176	0	0	0	0	103878	246723	16	56	19	9
4	3	100	21824	11088	13269140	0	0	0	0	48625	97288	15	16	9	60
11	2	100	20932	11080	13269808	0	0	0	0	110760	236774	14	41	24	20
1	12	100	23624	11084	13267488	0	0	0	204	69117	148611	15	27	25	33
7	5	100	24996	11096	13267476	0	0	0	164	24495	48552	13	10	30	48
1	12	100	20792	11096	13271872	0	0	0	0	25659	54331	8	9	26	56
6	8	100	21984	11080	13269920	0	0	0	20	46309	101404	16	18	51	15
4	9	100	22764	11080	13268956	0	0	16	0	88553	229557	17	35	38	11

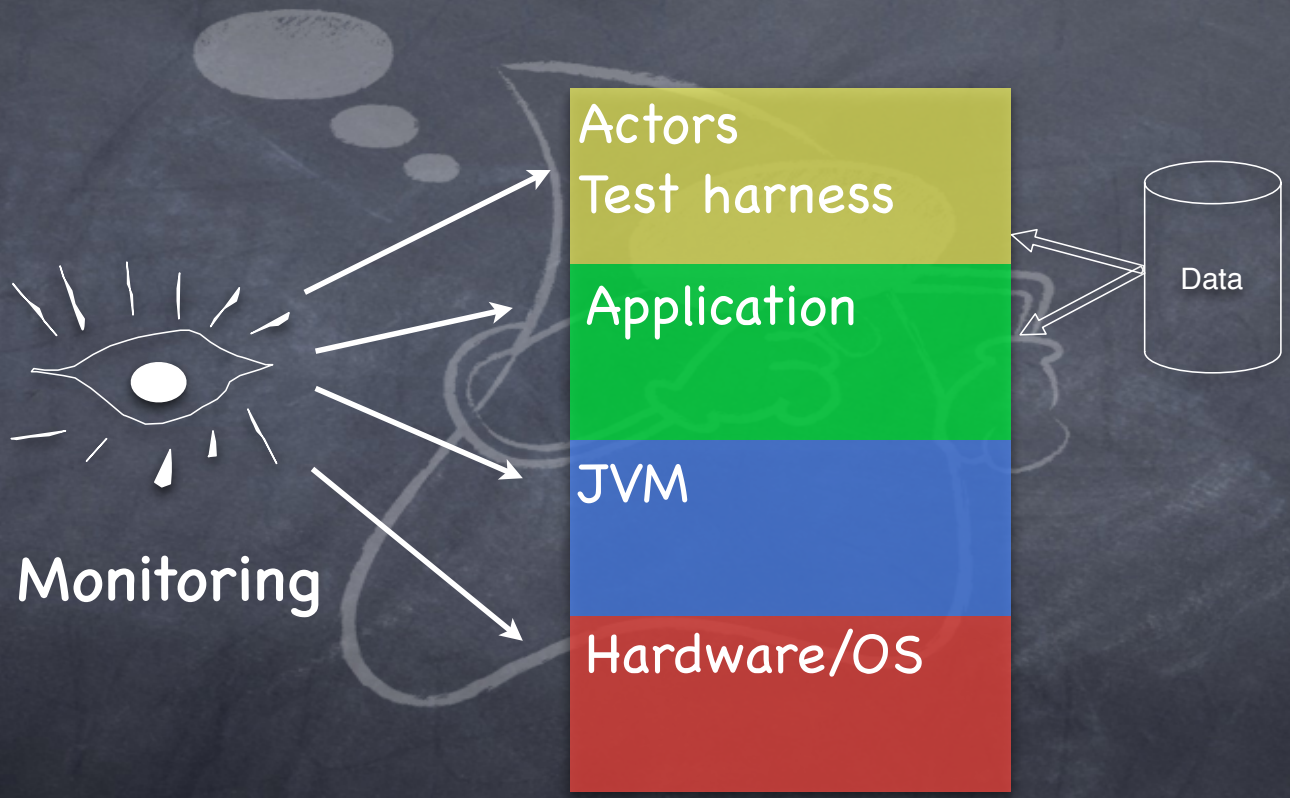
KODEWERK

Benchmarking Process

```
Benchmark benchmark = new Benchmark()
benchmark.configure();
performance = benchmark.baseline(application);
user.setHappy(performance.meets(requirements));
while (( ! user.isHappy()) && (user.hasMoney())) {
    Profiler profiler = performance.identifyDominatingConsumer();
    profilingResults = benchmark.profile(profiler);
    application.fixUsing( profilingResults);
    while ( application.failsQA())
        application.debug();
    performance = benchmark.baseline(application);
    user.setHappy(performance.meets(requirements));
}
```

KODEWERK

Things We Need

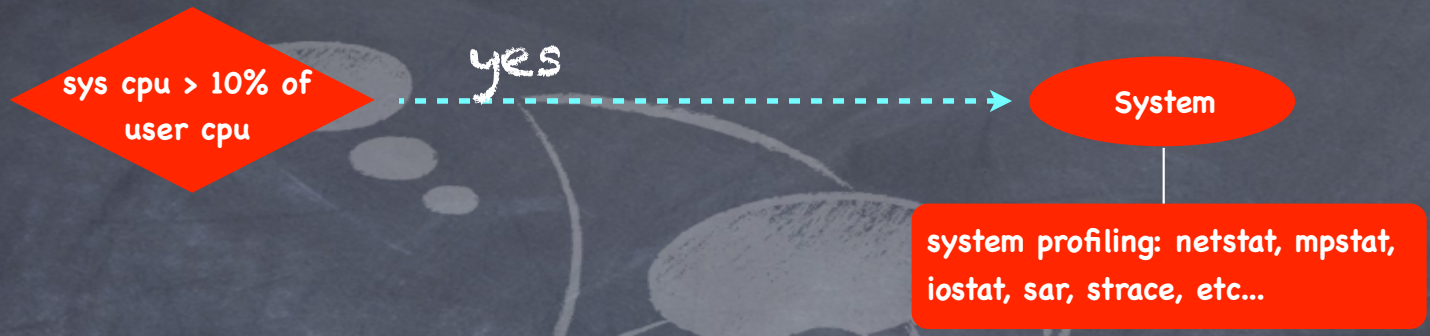


KODEWERK

Time for a demo

KODEWERK

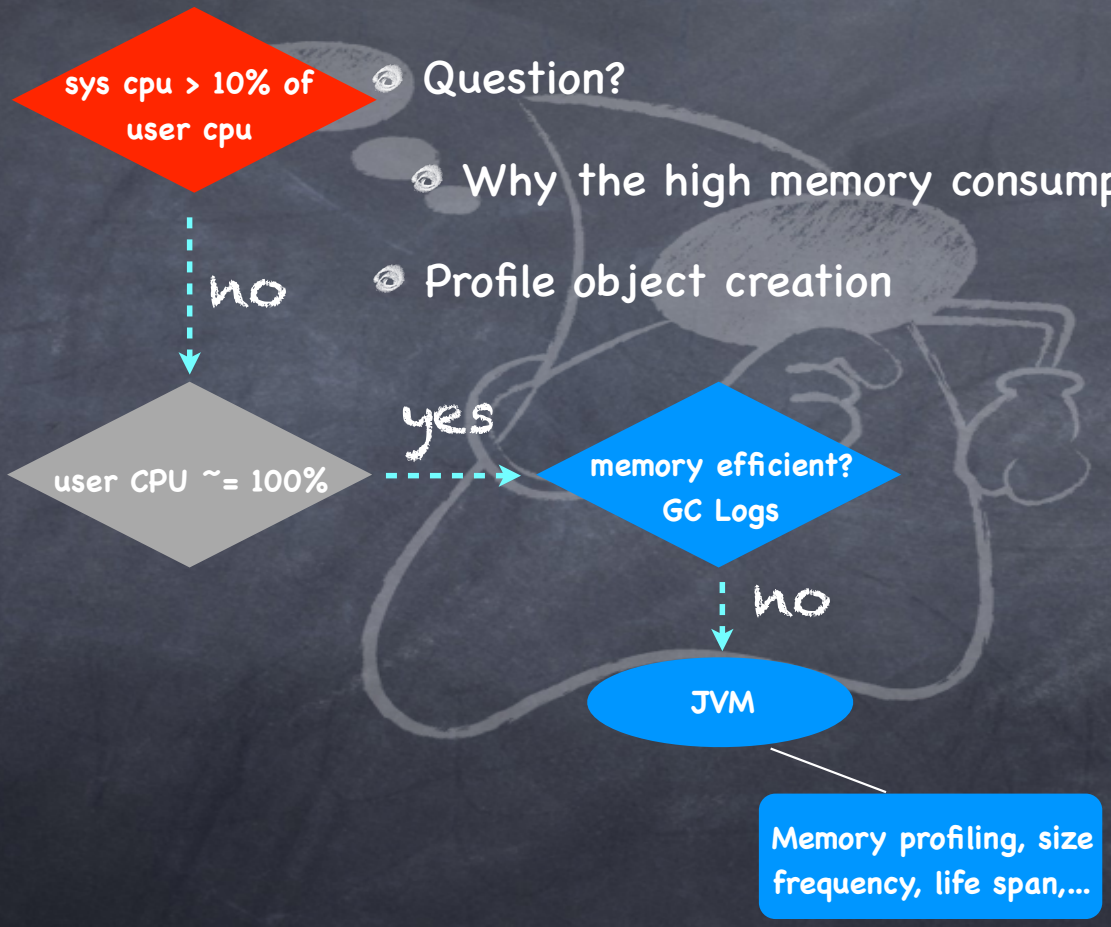
Dominating Consumer #1



- Question?
  - Why the high level of kernel CPU?
- Observations
  - no disk I/O, network activity or video
  - context switching due to lock contention?
- Monitor threads with VisualVM

KODEWERK

Dominating Consumer #2

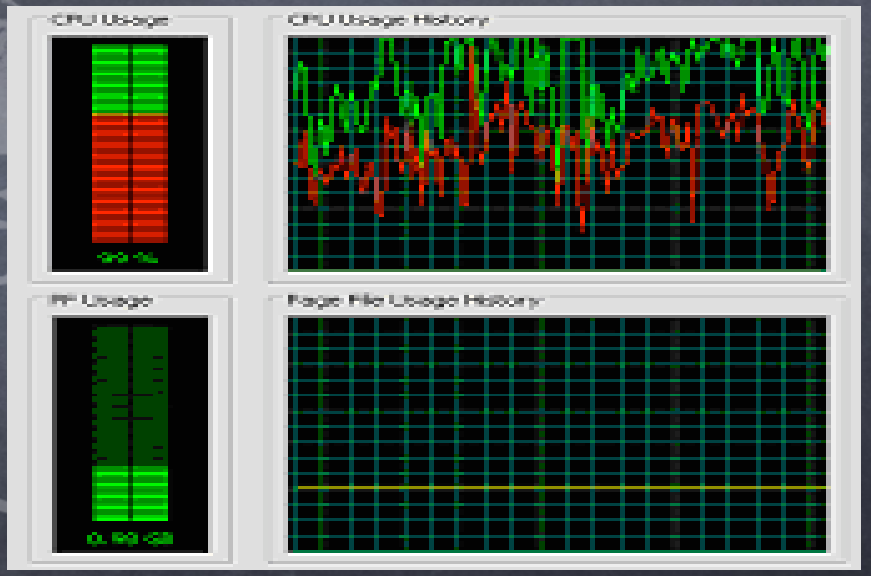


- Question?
- Why the high memory consumption
- Profile object creation

KODEWERK

What About Our Customer

- Need to gather clear requirements
- Develop a sound benchmarking environment
  - get better measurements
- Always identify dominating consumer
  - refocus teams on problems that matter



KODEWERK

Questions?