Breizh C@mp
Mix de technologies

Performance Pudding

Une recette simple et économique

#performancepudding

Vincent Beretti - @vberetti

Une recette

Performance pudding

Facile

Bon marché

- Proposée par Vincent Beretti
  - Architecte
  - @vberetti

IPPON
Enterprise Java Delivery

Au menu d’aujourd’hui

- des concepts
- des ingrédients
- des outils
- de l’inspiration

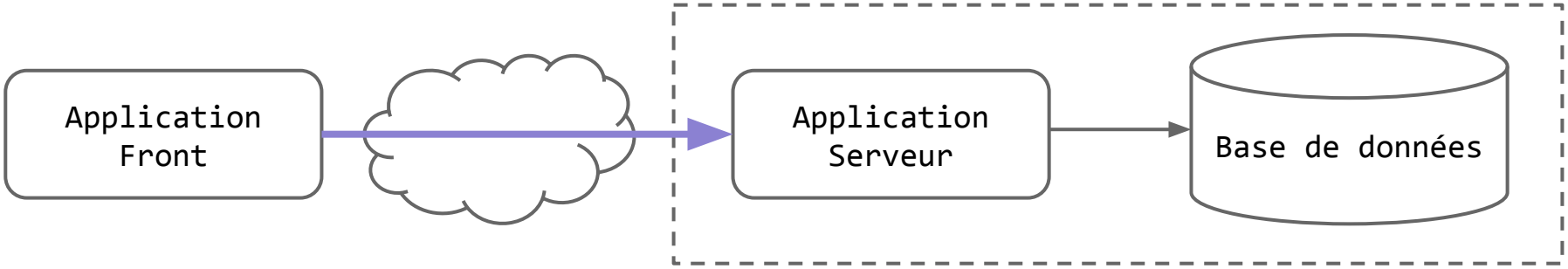
Plus précisement

- Ecosystème
- Système
- JVM
- Applicatifs
- Tirs de performance

Ecosystème

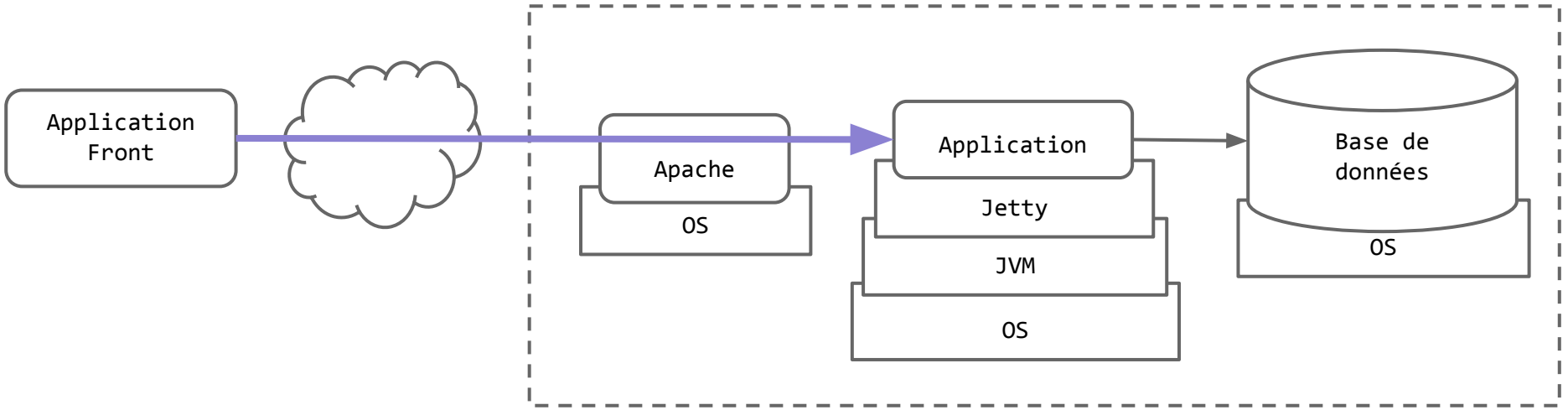
Ecosystème

- Bien connaître son système
  - Prenons une application :
    - Frontend en Angular
    - Backend en Java/JAX-RS
    - Base de données sur PostgreSQL
  - Les problèmes peuvent venir
    - du back
    - du front
    - de la base de données



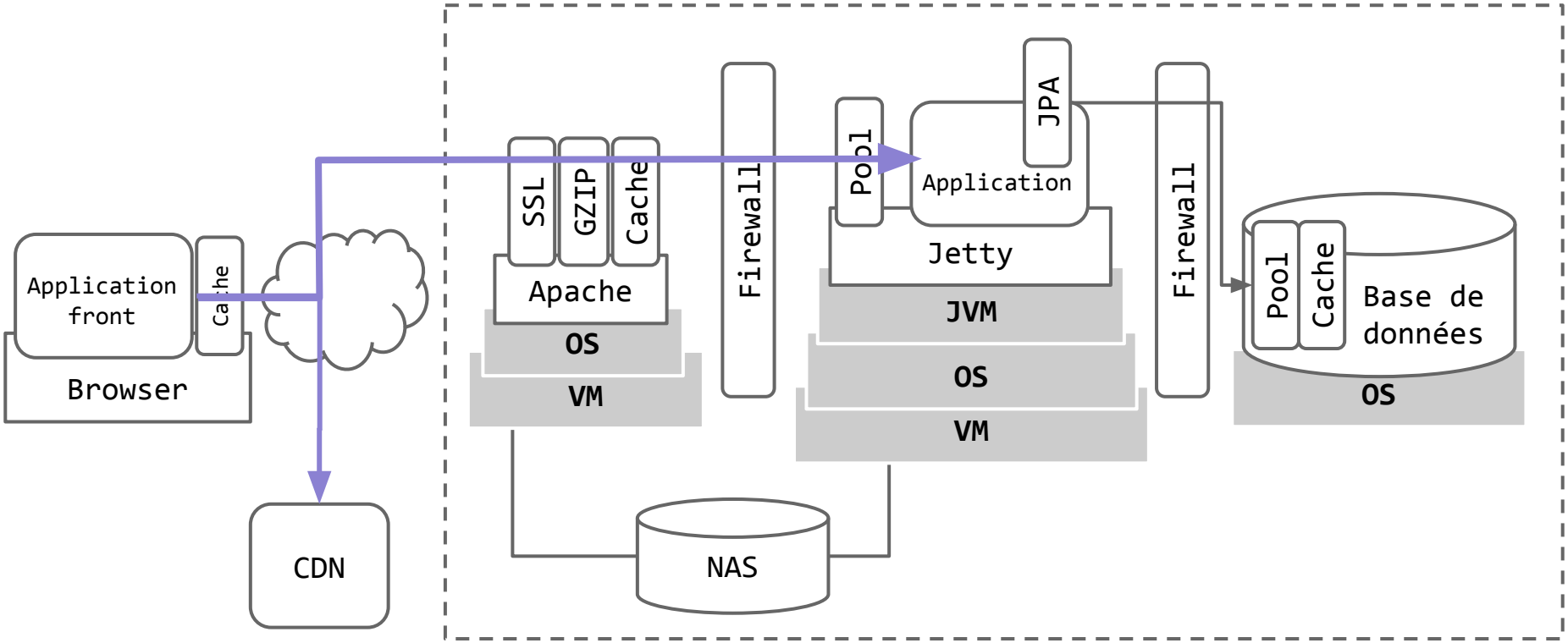
Ecosystème

- Les problèmes peuvent venir
  - du front
  - du back
  - de la base de données
  - de l'OS
  - de la JVM
  - de Jetty
  - de Apache



Ecosystème

- Les problèmes peuvent venir ...

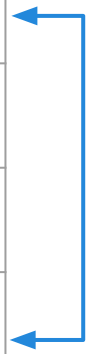


Une brève histoire du temps … ou pas

“Numbers Everyone Should Know” - Jeff Dean

Building Software Systems at Google and Lessons Learned

	Tps	Ratio	Analogie
L1 cache	0.5ns	1	1 s
Main Memory access	100ns	200	3 min
Send 1KB over Gbps network	10,000ns	20,000	5 h
Read seq 1MB from **Memory**	250,000ns	500,000	5 days
Round trip in same datacenter	500,000ns	1,000,000	11 days
Read seq 1MB from **SSD**	1,000,000ns	2,000,000	1 month
Read seq 1MB from **Disk**	20,000,000ns	40,000,000	1.5 years
Send Packet US -> Europe -> US	150,000,000ns	300,000,000	10 years



x 80 !

Système

Système

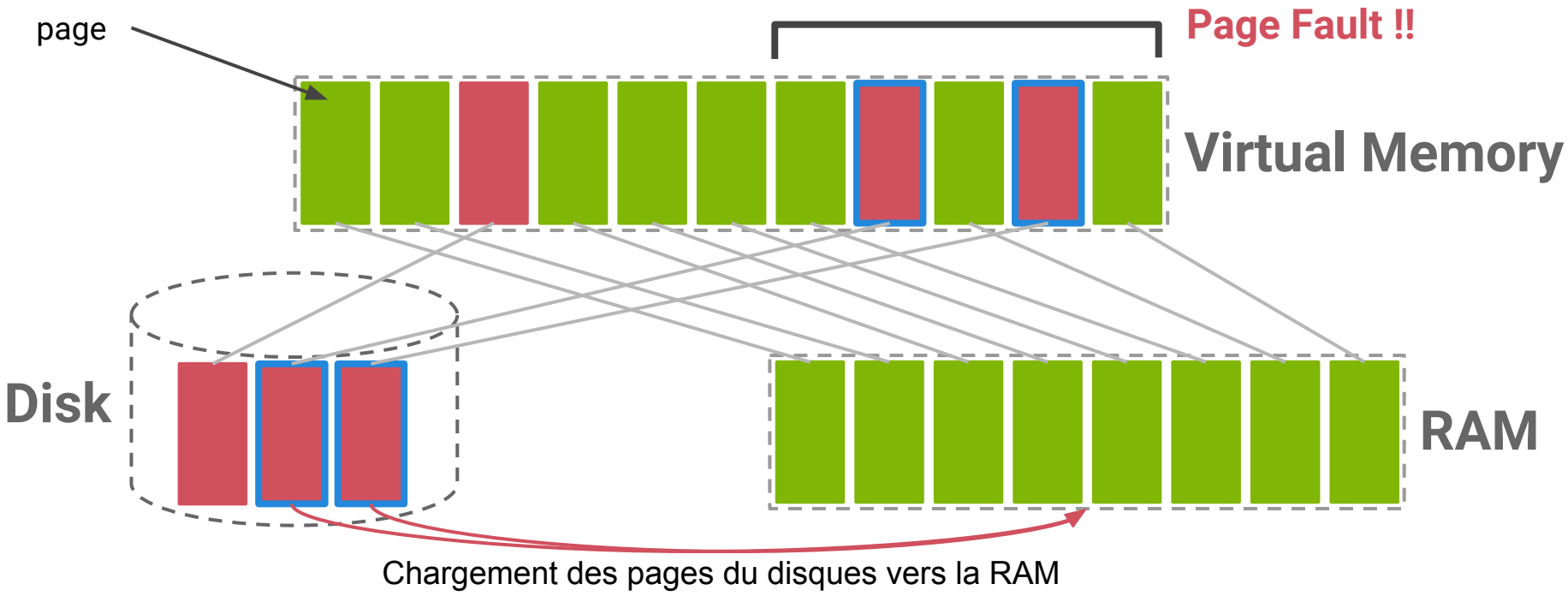
- Swap
- Virtual machine
- I/O
- Systat

Attention
ces slides ne sont pas des slides sur Windows$_{me}$

Merci de votre compréhension

Swap

Mécanisme d'extension de la mémoire en utilisant virtuellement une partie du disque



Ratio vitesse de lecture RAM/Disk : 1/80 !

Swap

```
> free -h
```

```
             total       used       free     shared    buffers     cached
Mem:          237M       218M        18M       228K       288K        20M
-/+ buffers/cache:       197M        39M
Swap:         2.0G       506M       1.5G
```

```
> vmstat 1
```

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0 271960  10964    168  27076   32 1696  4260  1720  892 1188  1  3 95  1  0
 2  0 271744   9864    168  27808  456    0  1120     0  630 1275  2  1 98  0  0
 0  0 271336   6092    176  30400  272    0  3336    12  737 1448  2  1 96  1  0
 0  0 272948   7148    172  32644 2172 3180 12632  3244 1642 2682  2  2 93  3  0
 0  0 272948   7700    172  32560    0    0     0     0  401  829  2  1 97  0  0
 0  0 272548   6704    172  32688  432    0   572     0  613 1274  3  1 96  0  0
 0  0 272540   6740    180  32700   64    0    64    12  351  756  1  0 98  0  0
```

si : Amount of memory swapped in from disk (/s)
so : Amount of memory swapped to disk (/s)

Swap

- Swappiness
  - représente la tendance à anticiper l'utilisation de Swap (0-100)
  - par défaut à 60
  - `> cat /proc/sys/vm/swappiness`

  - `swappiness` à 1
  - ajout de RAM

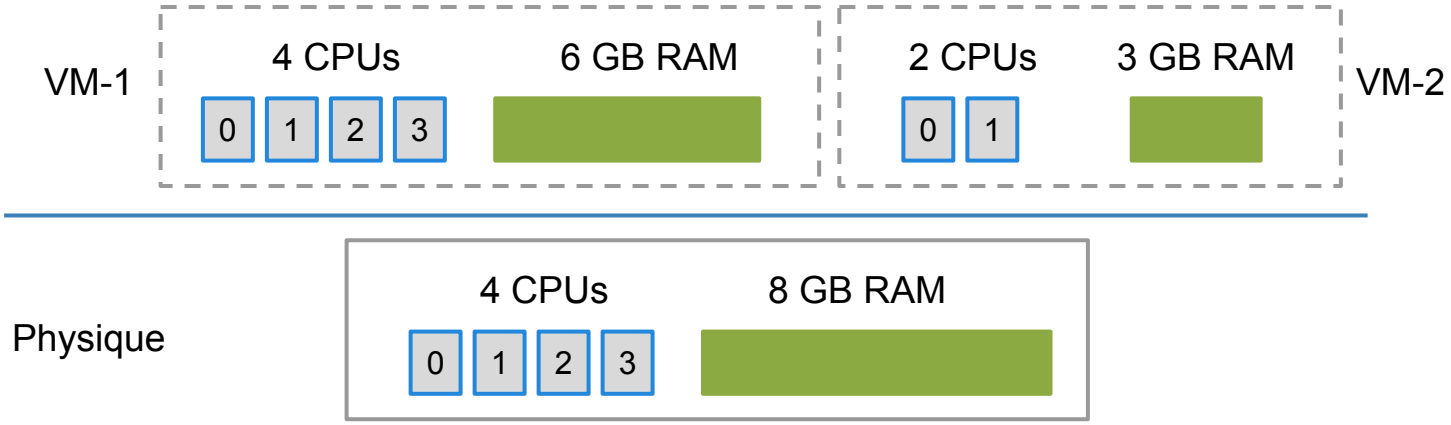
~~top~~ vmstat Chef

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0 271960  10964    168  27076   32 1696  4260  1720  892 1188  1  3 95  1  0
 2  0 271744   9864    168  27808  456    0  1120     0  630 1275  2  1 98  0  0
 0  0 271336   6092    176  30400  272    0  3336    12  737 1448  2  1 96  1  0
 0  0 272948   7148    172  32644 2172 3180 12632  3244 1642 2682  2  2 93  3  0
 0  0 272948   7700    172  32560    0    0     0     0  401  829  2  1 97  0  0
 0  0 272548   6704    172  32688  432    0   572     0  613 1274  3  1 96  0  0
 0  0 272540   6740    180  32700   64    0    64    12  351  756  1  0 98  0  0
 0  1 271840   8220    180  31304 1292    0  4716    40 1313 2127  2  2 94  1  0
 0  0 271184   6764    168  31728 1292  160 10356   164 1285 2188  3  2 94  2  0
 0  0 271184   6772    168  31764    0    0    44     0  444 1041  2  0 98  0  0
```

us	user	% utilisé par du code client
sy	system	% utilisé par le noyau
id	idle	% non-utilisé
wa	wait	% en attente d’I/O
st	steal	% en attente de l’hyperviseur

Steal : Cauchemar en VMs

Sur-allocation des resources



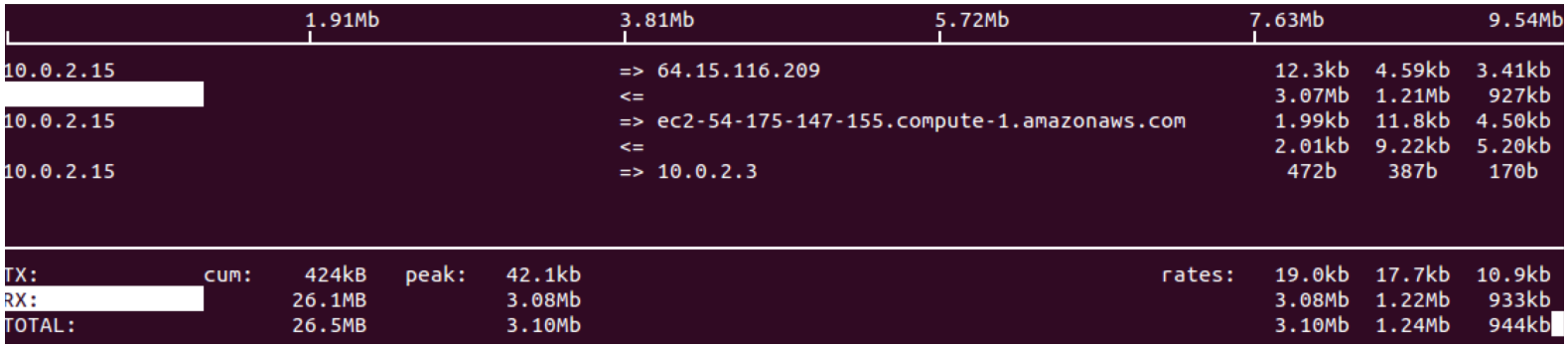
CPU : 4 + 2 != 4
RAM : 6 + 3 != 8

-> Read the F***ing VMWare's recipe

- Réservation Processeur
- [Reservation Mémoire]

I/O Wait

- Réseau
  - > sudo apt-get install iftop
  - > iftop



- Disques
  - > sudo apt-get install systat
  - > iostat -n

```
Device:            tps   kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda            6560.00   160512.00      5292.00     160512       5292
sdb               2.00        8.00         0.00          8          0
```

Egalement dans systat

- sar
  - mode interactif ou collecte
  - analyse post-mortem CPU, I/O, mémoire, …

- mpstat
  - liste l'activité par CPU
  - `> mpstat -A`

```
16:11:21     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal
16:11:21     all    2.70    0.04    4.81    7.28    0.00    0.91    0.00
16:11:21       0    2.69    0.06    5.24    8.56    0.01    3.73    0.00
16:11:21       1    2.71    0.04    5.05    7.26    0.00    0.01    0.00
16:11:21       2    2.73    0.03    4.92    6.78    0.00    0.01    0.00
16:11:21       3    2.68    0.03    4.08    6.58    0.00    0.00    0.00
```

JVM

JVM

- JIT - Just In Time
- Garbage Collector
- Thread Dump

Cuisson Just In Time

- JIT Compiler
  - traduit le bytecode en instructions processeur
  - et l'**optimise !**
- Oubliez les
  - *"Je vais mettre des final partout pour que la JVM optimise"*
  - *"Je vais inliner cette methode pour économiser un appel"*
  - *"Je vais ré-ordonner les conditions parce que celle la est plus utilisée"*
  - *"Je fais un benchmark en faisant tourner 100 fois la méthode"*

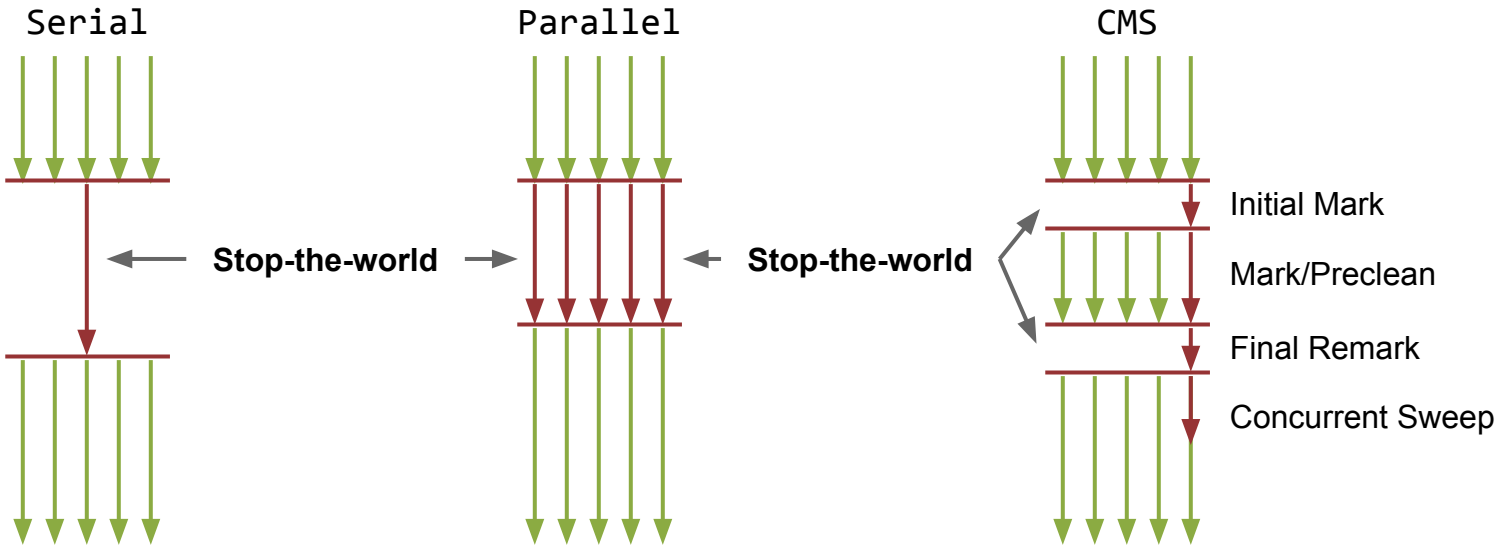
Cuisson Just In Time

- Optimiser ? le JIT le fera mieux que vous car :
  - il est plus intelligent
  - il le fait dynamiquement

- Optimisation
  - Statique : re-ordering, nettoyage, pre-compute, read cache, inlining, CHA, ...
  - Dynamique (au cours des executions)
    - optimise agressivement pour le chemin “courant”
    - dé-optimise pour s’adapter et ré-optimiser

Garbage Collector

- Garbage collector
  - Nettoyage automatique de la Heap
    - Old/New
  - Plusieurs algorithmes
    - throughput/low latency
  - [Java 8] par défaut Parallel (throughput)



Garbage Collector

- Minor GC
  - nettoyage New
- Major GC
  - Nettoyage Old

- Full GC
  - Quand ?
    - si une Major GC ne nettoie pas assez
    - si la heap est trop fragmentée
  - Tout se stoppe et grand nettoyage
  - Impact sur les performances si trop fréquent

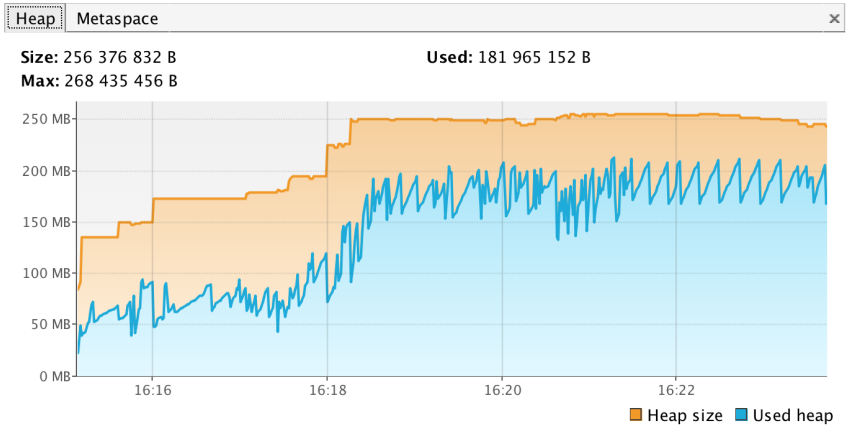
Garbage Collector

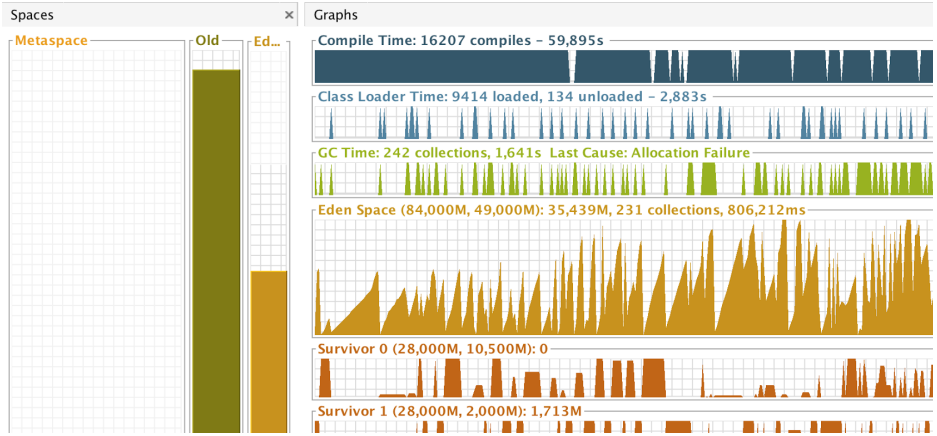
- Outils
  - Activer les logs du GC
  - *-XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xloggc:<file>*

```
.247: [GC (Allocation Failure) 16.247: [ParNew: 39296K->4298K(39296K), 0.0136164 secs] 105848K->74919K(1
.264: [GC (CMS Initial Mark) [1 CMS-initial-mark: 70620K(87424K)] 75030K(126720K), 0.0026111 secs] [Time
.267: [CMS-concurrent-mark-start]
.387: [CMS-concurrent-mark: 0.120/0.120 secs] [Times: user=0.54 sys=0.03, real=0.12 secs]
.388: [CMS-concurrent-preclean-start]
.401: [CMS-concurrent-preclean: 0.012/0.013 secs] [Times: user=0.06 sys=0.01, real=0.01 secs]
.401: [CMS-concurrent-abortable-preclean-start]
.058: [GC (Allocation Failure) 17.058: [ParNew: 39242K->2403K(39296K), 0.0081545 secs] 109863K->73023K(1
.670: [CMS-concurrent-abortable-preclean: 1.237/1.269 secs] [Times: user=4.68 sys=0.28, real=1.27 secs]
.682: [GC (CMS Final Remark) [YG occupancy: 28170 K (39296 K)]17.682: [Rescan (parallel) , 0.0170772 sec
ub sy ol table, 0.0253866 secs]17.768: [scrub string table, 0.0022623 secs][1 CMS-remark: 70620K(87424K
.774: [CMS-concurrent-sweep-start]
.869: [CMS-concurrent-sweep: 0.095/0.095 secs] [Times: user=0.37 sys=0.03, real=0.10 secs]
.869: [CMS-concurrent-reset-start]
.869: [CMS-concurrent-reset: 0.000/0.000 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
.992: [GC (Allocation Failure) 17.992: [ParNew: 37347K->2789K(39296K), 0.0080808 secs] 105543K->70986K(1
.851: [GC (Allocation Failure) 18.851: [ParNew: 37733K->2966K(39296K), 0.0067131 secs] 105930K->71162K(1
.720: [GC (Allocation Failure) 19.720: [ParNew: 37910K->3052K(39296K), 0.0070252 secs] 106106K->71249K(1
.584: [GC (Allocation Failure) 20.585: [ParNew: 37996K->3388K(39296K), 0.0060825 secs] 106193K->71585K(1
```

Garbage Collector

- Au besoin `$JDK_HOME/bin`
  - JVisualVM + plugin VisualGC





  - `> jstat -gccause <vmid> 1s`

S0	S1	E	O	M	CCS	YGC	YGCT	FGC	FGCT	GCT	LGCC	GCC
0,00	0,00	69,96	44,22	93,13	86,88	15	0,051	3	0,217	0,268	Allocation Failure	No GC
0,00	0,00	82,75	44,22	93,13	86,88	15	0,051	3	0,217	0,268	Allocation Failure	No GC
0,00	0,00	87,82	44,22	93,13	86,88	15	0,051	3	0,217	0,268	Allocation Failure	No GC
90,92	0,00	100,00	62,28	93,13	86,88	16	0,051	3	0,217	0,268	Allocation Failure	Allocation Failure
91,45	0,00	21,55	62,28	93,11	87,62	16	0,058	3	0,217	0,275	Allocation Failure	No GC
91,45	0,00	23,34	62,28	93,11	87,62	16	0,058	3	0,217	0,275	Allocation Failure	No GC
91,45	0,00	26,44	62,28	93,11	87,62	16	0,058	3	0,217	0,275	Allocation Failure	No GC
91,45	0,00	28,32	62,28	93,11	87,62	16	0,058	3	0,217	0,275	Allocation Failure	No GC
91,45	0,00	28,89	62,28	93,11	87,62	16	0,058	3	0,217	0,275	Allocation Failure	No GC
91,45	0,00	30,99	62,28	93,11	87,62	16	0,058	3	0,217	0,275	Allocation Failure	No GC

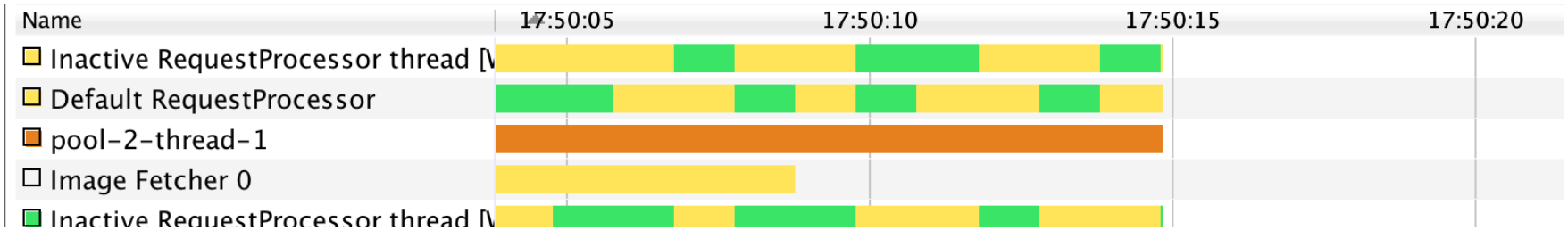
Thread Dump

- Threads avec leur statut et leur stacktrace
  - Brut (`jstack` / `kill -3`)

```
"Java2D Disposer" #20 daemon prio=10 os_prio=31 tid=0x00007fe63503d000 nid=0xde0f in Object.wait() [0x00000001284c8000]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:143)
        - locked <0x00000007b03a3400> (a java.lang.ref.ReferenceQueue$Lock)
        at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:164)
        at sun.java2d.Disposer.run(Disposer.java:148)
        at java.lang.Thread.run(Thread.java:745)

"Java2D Queue Flusher" #19 daemon prio=10 os_prio=31 tid=0x00007fe6330dd000 nid=0xd907 in Object.wait() [0x000000012772a000]
   java.lang.Thread.State: TIMED_WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        at sun.java2d.opengl.OGLRenderQueue$QueueFlusher.run(OGLRenderQueue.java:203)
        - locked <0x00000007b03c2c60> (a sun.java2d.opengl.OGLRenderQueue$QueueFlusher)
```

  - En visuel (`jvisualvm`, `jconsole`, `jmc`)



Thread Dump

- Status des Threads
  - RUNNABLE
  - WAITING
  - BLOCKED
- Exemple d'utilisation : repèrer une contention

```
"BLOCKED_TEST pool-1-thread-1" prio=6 tid=0x0000000006904800 nid=0x28f4 runnable [0x00785f000]
   java.lang.Thread.State: RUNNABLE
        at java.io.FileOutputStream.writeBytes(Native Method)
        ...
        - locked <0x0000000780a000b0> (a com.nbp.theplatform.threaddump.ThreadBlockedState)
        at com.nbp.theplatform.threaddump.ThreadBlockedState$1.run(ThreadBlockedState.java:7)
        at java.lang.Thread.run(Thread.java:662)

"BLOCKED_TEST pool-1-thread-2" prio=6 tid=0x0000000007673800 nid=0x260c waiting for monitor entry [0x008abf000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at com.nbp.theplatform.threaddump.ThreadBlockedState.monitorLock(ThreadBlockedState.java:43)
        - waiting to lock <0x0000000780a000b0> (a com.nbp.theplatform.threaddump.ThreadBlockedState)
        ...
```

Applicatif

Applicatif

Synchronized

static Maps

Thread pools

Tx Serializable isolation level

logger.debug()

N+1 SELECT

Pagination

Caches

Session timeout

@OneToMany(fetch=FetchType.EAGER)

Tx Requires New propagation

Applicatif

Synchronized

static Maps

Thread pools

logger.deb

vel

Caches

- Monitorer le hit ratio du cache
  - *hit ratio = hits / (hits + misses)*
- Un hit ratio trop faible peut impacter négativement la mémoire

ut

@OneToMany(fetch=FetchType.EAGER)

Tx Requires New propagation

Tirs de performances

Junk Food Benches

“Prends une VM à 1 CPU, on divisera par 4 les temps de réponse”

“Attends, je préchauffe l’application en affichant la homepage”

“Les resultats des tirs ? Le temps médian est de 200ms”

“Je ne sais pas ce qu’il s’est passé, on ne fait juste des tests black-box !”

“Ils veulent que le site réponde en moins d’une seconde”

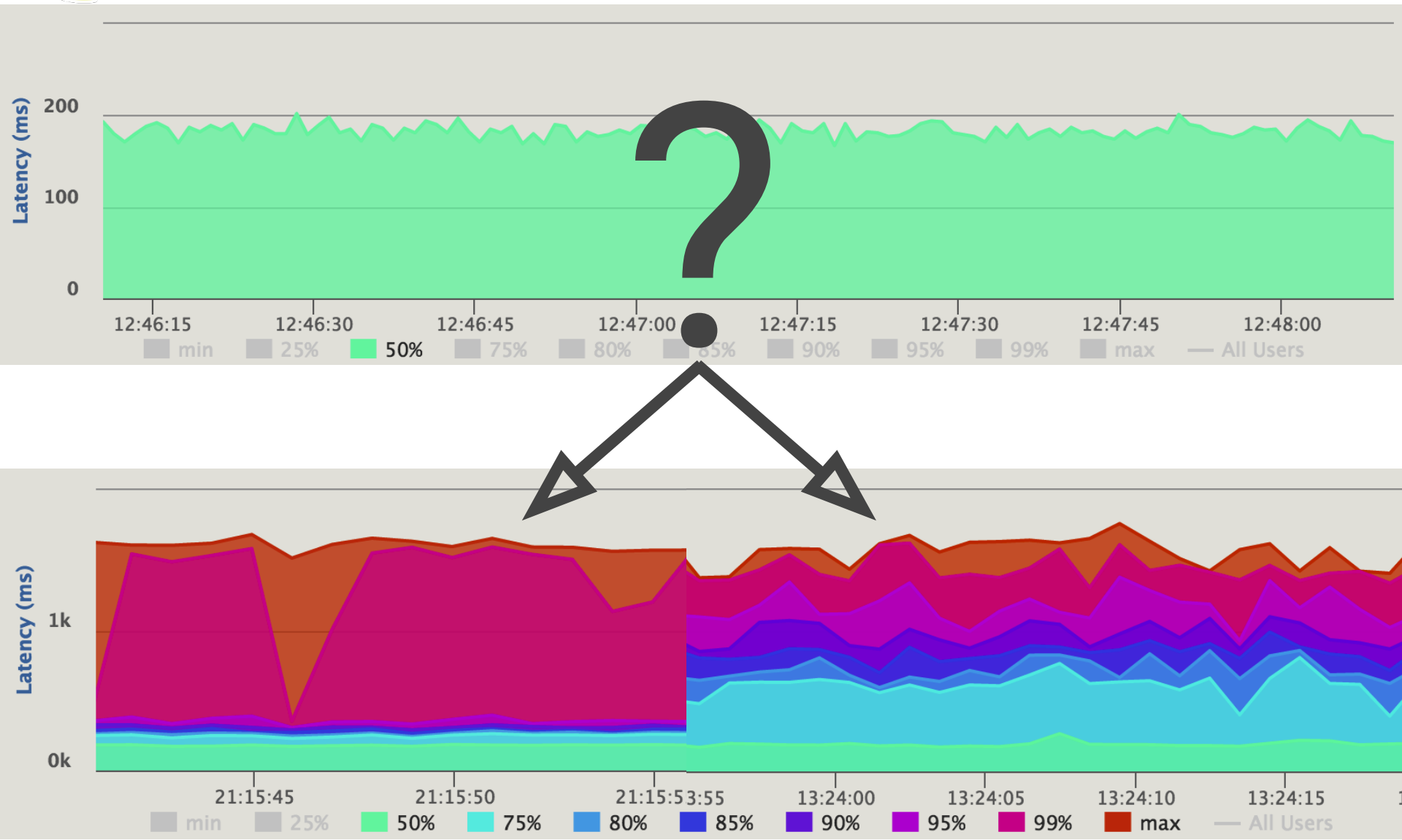
...

Médiane et percentiles

- Médiane
  - la moitié des requêtes a un temps de réponse inférieur ou égal
  - et l'autre moitié ?
  - et le max ?
- Percentiles
  - 50%ile == médiane
  - 99%ile seul 1% des requêtes sont plus lentes
  - 99.99%ile seul 0.01% des requêtes sont plus lentes

Mesurer un eventail de percentiles (75%, 90%, …)

Percentiles



Seuil de performances

- Comprendre l'impact fonctionnel de la définition d'un seuil de performance

Risque pour un utilisateur unique d'avoir pendant sa session un temps de réponse supérieur à un percentile donné

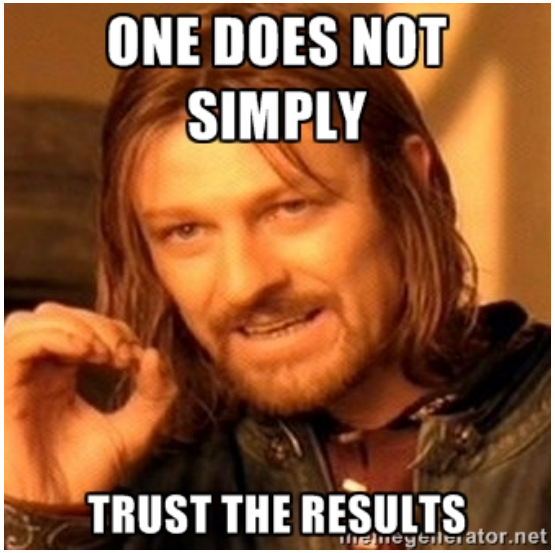
Requetes/session	50%ile	95%ile	99%ile	99.9%ile	99.999%ile
25	+100%	+100%	25%	2.5%	0.0025%
50	+100%	+100%	50%	5%	0.05%
100	+100%	+100%	100%	10%	0.1%

- Définir le niveau d'exigence par rapport à un coût de mise en oeuvre

En bref


PERFORMANCE,
PERFORMANCE EVERYWHERE
imgflip.com


MESUREZ
MESUREZ MESUREZ MESUREZ


ONE DOES NOT
SIMPLY
TRUST THE RESULTS


WHAT'S IN THE BLACK BOX
PERFORMANCE TEST ?
memegenerator.net

Questions

?

```
scenario("PerformancePudding")
    .during(5 minutes) {
        exec(http("questions").get("/questions"))
            .check(status.not(504))
    }
```

Références

Pointeurs

- https://www.parleys.com/tutorial/priming-java-speed
- https://www.parleys.com/tutorial/how-i-not-i-measure-latency

Crédit Photos

- https://flic.kr/p/ofMfMs
- https://flic.kr/p/7S6smv
- http://commons.wikimedia.org/wiki/File:Plum_pudding.jpg