Scaling the tech up

Best Practices, Architecture, Distributed Systems…

Disclaimer

Hi, I’m Julien

My experience

Streaming, Serverless Computing, PubSub

What we will cover:

Some basic about distributed systems and scalability

Practice: how we can build a scalable social network

What we will NOT cover but are equally important:

Scaling the organization

Scaling the tools

Management of the technical debt at Scale

Distributed Systems

What is it? Is it the cloud (lol)?

Avoid distributing your app as long as you can

Let’s define what “scale” means in this talk:

The ability to handle more demand!

While maintaining or improving throughput and latency

While maintaining or improving availability and durability. (MeanTimeToRecover should not increase significantly with system size.)

Using infrastructure in a cost-efficient manner: i.e. by adding hardware resources approximately linear in the increase in amount of work

With marginal increase in costs for human operators.

Scaling up Software

- Is not about using techno, or cloud, or whatever
- It’s not free
- It’s not magical
- It’s distributed systems and data structures and it’s engineering excellence, processes and methods.

- **Linus Torvald's** comment: I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his **data structures** more important. Bad programmers worry about the code. Good programmers worry about **data structures** and their relationships.

How to scale, the short answer

- Best DS for best operation space/time complexity
- Distribute those DS
- Horizontally scale those distributed DS

- Be careful to:
  - Blast radius
  - Fault tolerance
  - Outages
  - Partitioning (%)

How to scale, the long answer

- It’s a journey:
  - Engineering rigors (code reviews, tests, CM…)
  - CICD, Beta, Gamma…
  - Granularity
  - Ops readiness (logs, metrics, alarms) / DevOps / Load Tests
  - Reliable Programming, Data Structures, Define limits
  - PaaS?
  - Distributed Systems (%, Blast Radius)
  - And also: tech debt management, scale up tools, scale up organization…

Effective method:

- Hope is not a strategy
- Use systemic approach, methods and mechanisms

Effective methods (example)

1. Go back to the WHY: understanding the business is YOUR business
2. Define precisely what the system should do
   - TLA+!!!
3. Define limits of your systems
4. Use “recovery-oriented computing” at every level
5. Build strong isolation mechanisms to limit blast-radius of problems
6. Prepare for change:

”Definition of Software Architecture: a framework for the disciplined introduction of change" – Tom Demarco

- because: change is inevitable, and without a framework for change, each change results in hacks, and hacks significantly increase the probability of bugs, operator errors, worsening blast-radius, thrash, …

Recovery-oriented Computing

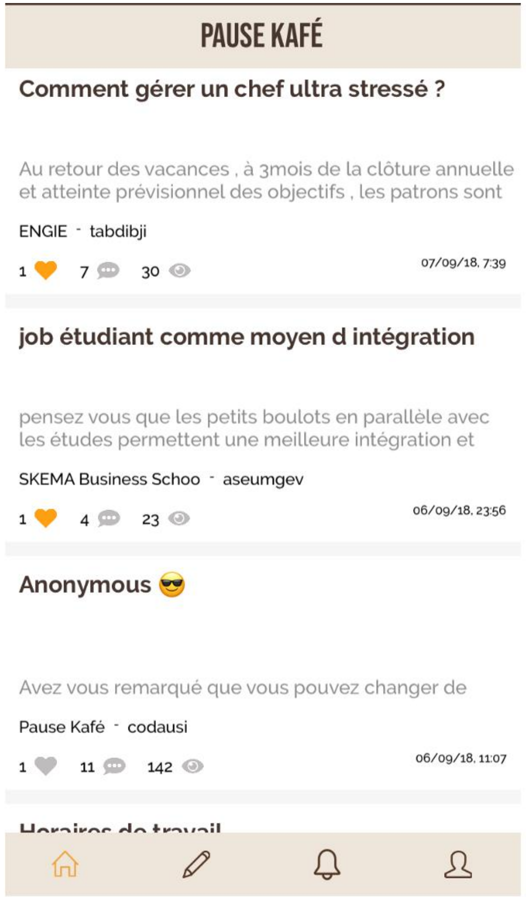
- Improve availability by **recovering very quickly** (which you can test), rather than trying to never fail (which is not feasible)

- Aggressively optimize time-to-recover, by:
    - quickly detecting the effects of bugs
    - quickly aborting/killing the smallest “unit of isolation” affected by the bug
    - optimizing the time required to restart each unit of isolation (aka. 'micro-reboots’)

- Make your system recover automatically

Let’s try it!

Let’s create a social network: pause-kafe.com



PAUSE KAFÉ

Comment gérer un chef ultra stressé ?

Au retour des vacances , à 3mois de la clôture annuelle et atteinte prévisionnel des objectifs , les patrons sont

ENGIE - tabdibji

1 ♥ 7 💬 30 👁 07/09/18, 7:39

job étudiant comme moyen d intégration

pensez vous que les petits boulots en parallèle avec les études permettent une meilleure intégration et

SKEMA Business Schoo - aseumgev

1 ♥ 4 💬 23 👁 06/09/18, 23:56

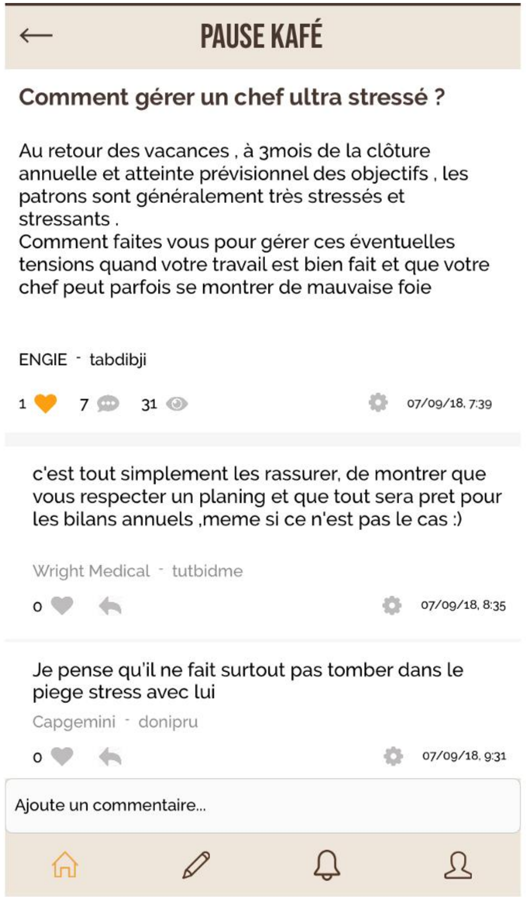
Anonymous 😎

Avez vous remarqué que vous pouvez changer de

Pause Kafé - codausi

1 ♥ 11 💬 142 👁 06/09/18, 11:07

Horaires de travail



← PAUSE KAFÉ

Comment gérer un chef ultra stressé ?

Au retour des vacances , à 3mois de la clôture annuelle et atteinte prévisionnel des objectifs , les patrons sont généralement très stressés et stressants .
Comment faites vous pour gérer ces éventuelles tensions quand votre travail est bien fait et que votre chef peut parfois se montrer de mauvaise foie

ENGIE - tabdibji

1 ♥ 7 💬 31 👁 07/09/18, 7:39

c'est tout simplement les rassurer, de montrer que vous respecter un planing et que tout sera pret pour les bilans annuels ,meme si ce n'est pas le cas :)

Wright Medical - tutbidme

0 ♥ 07/09/18, 8:35

Je pense qu'il ne fait surtout pas tomber dans le piege stress avec lui

Capgemini - donipru

0 ♥ 07/09/18, 9:31

Ajoute un commentaire...

Data Plane vs Control Plane

What do we need?

- A feed of posts, like facebook that people can comment. Those comments can also be commented

But WHY?

- Because it’s a new project, new startup, we hope to make a new social network for professional
- We are trying to prove the market

But WHY?

- Market not proven yet? Take technical debt
- The “Slack” solution; coding isn’t always the answer

Define the limits

- Ok now you proved the market and want to do your own dataplane?
- What's the scale you expect?
  - Market size
  - Market share over time (translate into TPS, # of customers…)
  - Size maximum of messages?
  - Review the UX: a huge amount of topic displayed in one shot isn't good

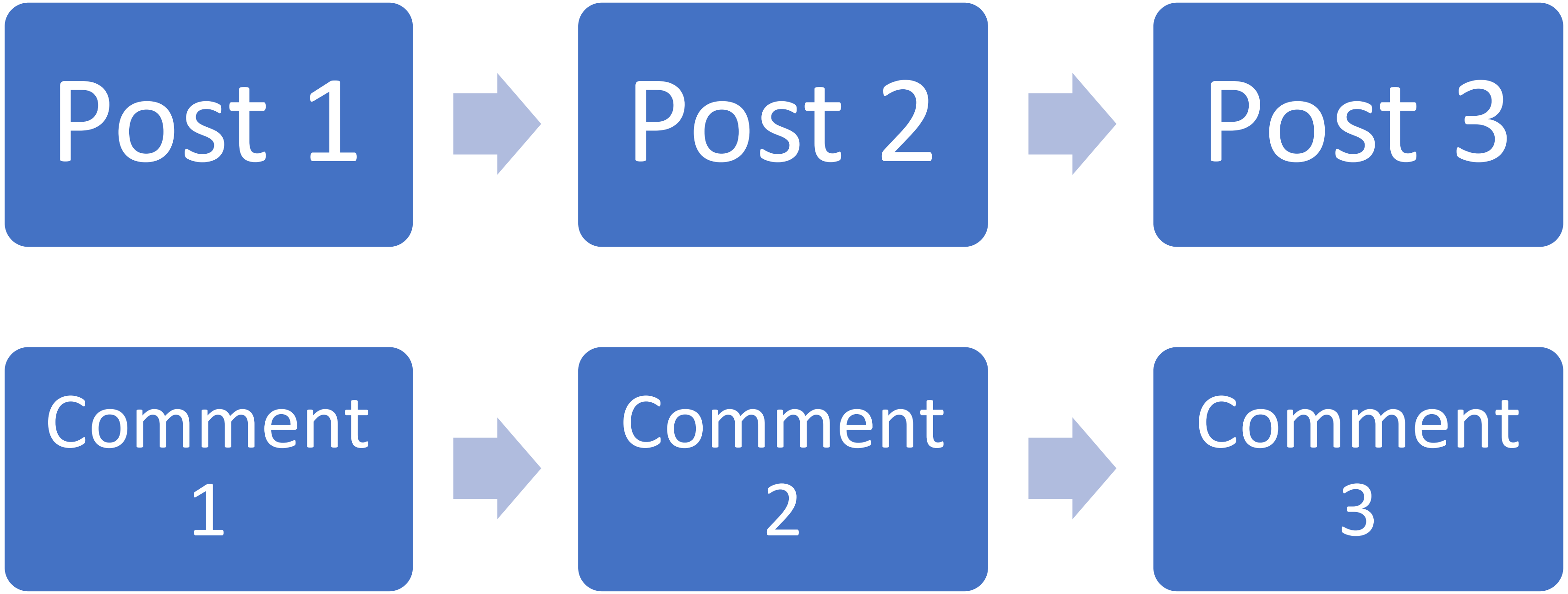
- Size of the team? Not much $ in bank? Time before company dies? (=low ops?)

- What are the properties of the system (liveness, safety property, non functional):
  - Best ordering
  - Durable
  - Low latency
  - Highly available
  - Can Infinitely grow

What’s the best data structure?

- Operation we will do the most?
  - Get Last Topics
  - Add new Topic
  - Get Next Topics
  - Best time / size complexity? -> Linkedlist!

What’s the best data structure?



Post 1

Post 2

Post 3

Comment 1

Comment 2

Comment 3

So we need a Linkedlist that can be:

- Low in ops. Most likely need a PaaS
- Distributed
- Horizontally scalable
- Replicated (fault tolerance, availability and co)
- Stored on disk
- Dead-lock free
- Space/cost optimized

Hashtable: the mother of all

- With an hashtable you can implement all DS in the world (in mostly the good ways)
- A good distributed hashtable – cloud scale ready will solve most the technical challenges
- Basic operations we need:
    - CAS !!!
    - Atomic append
    - Batch write and read
    - Atomic updates with conditional writes
    - Increments without read
    - Etc…

Hashtable: the mother of all

- OCI Object Store MD, DynamoDB, CosmosDB etc… allows to store key value efficiently with strong consistency and consistent partitioning on the primary key.

- We have the need to represents the data in form of a linked-list, and we want to benefits from the same guarantee as those KeyValueDB (availability, consistency, latency…).

- So let’s build a facade on top that allow us to have our Linkedlist

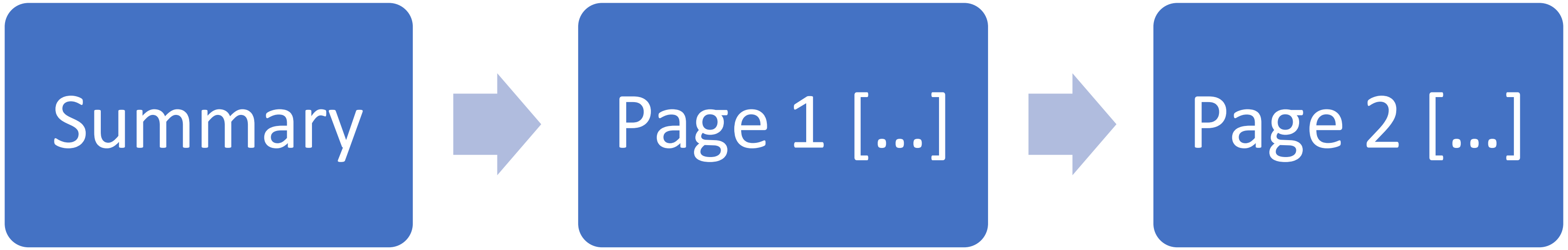
- A linked list is represented by multiple “page”.

- The first page of the LinkedList will contain a pointer to the current last “page” of the linkedlist (where new items should be appended), it is called the “summary” page.

- The next pages, also called “data page” contains the data appended to the linkedlist. Each page can contain an array of item and can store between 0 to N item.

- The goal of putting together those items in the same node is to not have to create a new element for each writes. Why? because we want to avoid to have to update the pointer in the summary page too often. This avoid doing too much random access write on the summary page, which on top of being slower, will be a “hotspot” prone to bug at scale. N threads using the linkedlist could increment the page summary in the same time which would lead to a lot of “blank” page.

Our distributed Linked List



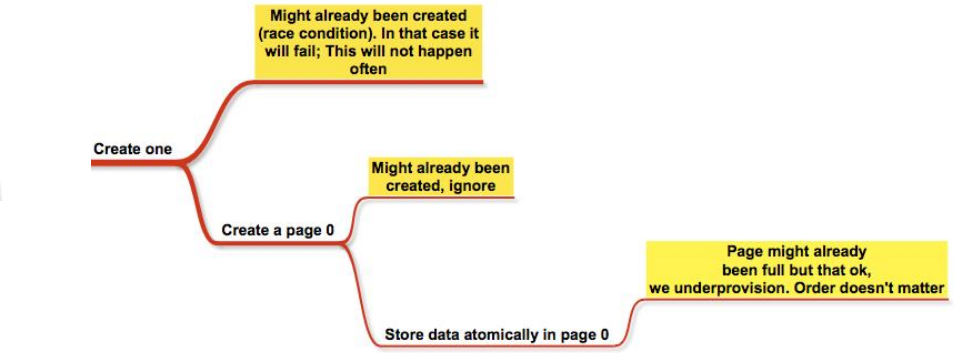
Summary

Page 1 […]

Page 2 […]

Recovery programming

- What wrong could happen when append data for example ? (bug, race conditions etc)



Create one

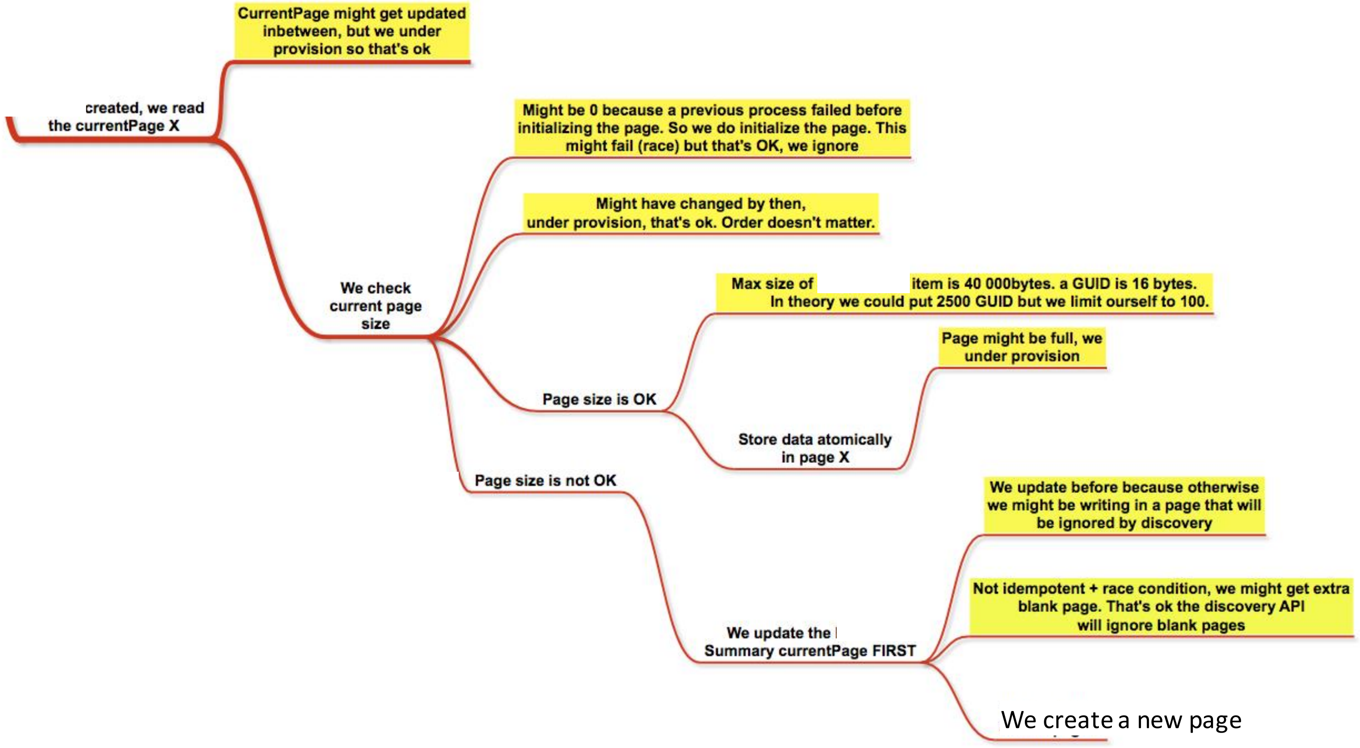
Might already been created (race condition). In that case it will fail; This will not happen often

Create a page 0

Might already been created, ignore

Store data atomically in page 0

Page might already been full but that ok, we underprovision. Order doesn't matter



created, we read the currentPage X

CurrentPage might get updated inbetween, but we under provision so that's ok

We check current page size

Might be 0 because a previous process failed before initializing the page. So we do initialize the page. This might fail (race) but that's OK, we ignore

Might have changed by then, under provision, that's ok. Order doesn't matter.

Page size is OK

Max size of item is 40 000bytes. a GUID is 16 bytes. In theory we could put 2500 GUID but we limit ourself to 100.

Store data atomically in page X

Page might be full, we under provision

Page size is not OK

We update the Summary currentPage FIRST

We update before because otherwise we might be writing in a page that will be ignored by discovery

Not idempotent + race condition, we might get extra blank page. That's ok the discovery API will ignore blank pages

We create a new page

Some code

```
/* This function is to optimize how much a data can be added by “page”.
If page == 1 (which is the minimum) it will create a new keyvaluestoredb item for each node in the linkedlist.
This will most likely not react well at scale (empty page, pointer incremented too much…).

The goal is to put as much data as possible in a single page but:
- The size should not go being the maximum keyvaluestore item size (which is 400 KB)
- BulkRetrieve in keyvaluestore is limited to 16 000KB. You will most likely want to retrieve a linkedlist in bulk, so keep this in mind.
- Each page contains 1KB of metadata
- Concurrent threads can be trying to add in the same page despite the limit being reached! It should be small window of few seconds when this happen, but my advice would be to over provisioning by assuming the maximum size of the keyvaluestore DB item is divided by 2 (and then tune things up or down depending on the scale)
- Overall, the number you put here, is the maximum of TPS for this instance of the linkedlist (req in the average same seconds) you can have before any of your customers get TooBigItemException

So for example, if the maximum size of value is 1 000 characters, in UTF-8, it mean each element maximum 1,074 bytes (1kb).
The maximum size of a page should be: 1KB + 1kb * N <= 400KB, N should be less than 390ish.
Then we apply the over-provisioning factor of /2, so we limit ourself to 200 elements.
Note: it would limit to retrieve 80 linked-list page in a BulkRetrieve operation */
module.exports.configureMaximumNumberOfElementPerPage = function(numberOfMaximumElement) {
        if (numberOfMaximumElement) {
                config.maxElementPerPage = numberOfMaximumElement;
        }
}
```

```
/* This will create the first summary page. This operation is idempotent. */
module.exports.idempotentCreate = function(id, metadata) {
    const summaryObject = defaultPageSummary(id, metadata);

    const itemInfo = {
    Item: summaryObject,
    Condition: ‘!summaryObject.id', //CAS on the id for idempotency
    };

    return keyvaluestoreDb.put(itemInfo).promise().then(res => summaryObject);
}
```

```
/*
If the page does not exists it first atomically increment the pointer to point to the “current” page.
This pointer might be incremented as the same time as another thread which is why you can have “blank” page that
you should ignore.
It then creates the new page by using the CAS mechanism. If the page get has been created in between, it just fails
silently.
Then it finally append the value (or directly append if the page was already created) by inserting the data into the
page atomically.
This operation is not idempotent and could insert duplicates due to retries.
Note: value needs to be an object
*/
module.exports.atomicAppend = function(id, value) {
	return getCurrentPage(id).then(function(currentPage) {
		return atomicAppendImpl(id, currentPage, value);
	});
}
```

```
 /* It will retrieve the top N most recent item that has been appended to the linkedlist.
 Be careful to bufferoverflow here. Avoid asking for a crazy amount */
 module.exports.retrieveLastMostRecent = function(id, numberOfItems, callback) {
         getCurrentPage(id)
         .then(function(currentPage) {
                 retrieveNElement(id, currentPage, null, numberOfItems, callback);
         });
 }
```

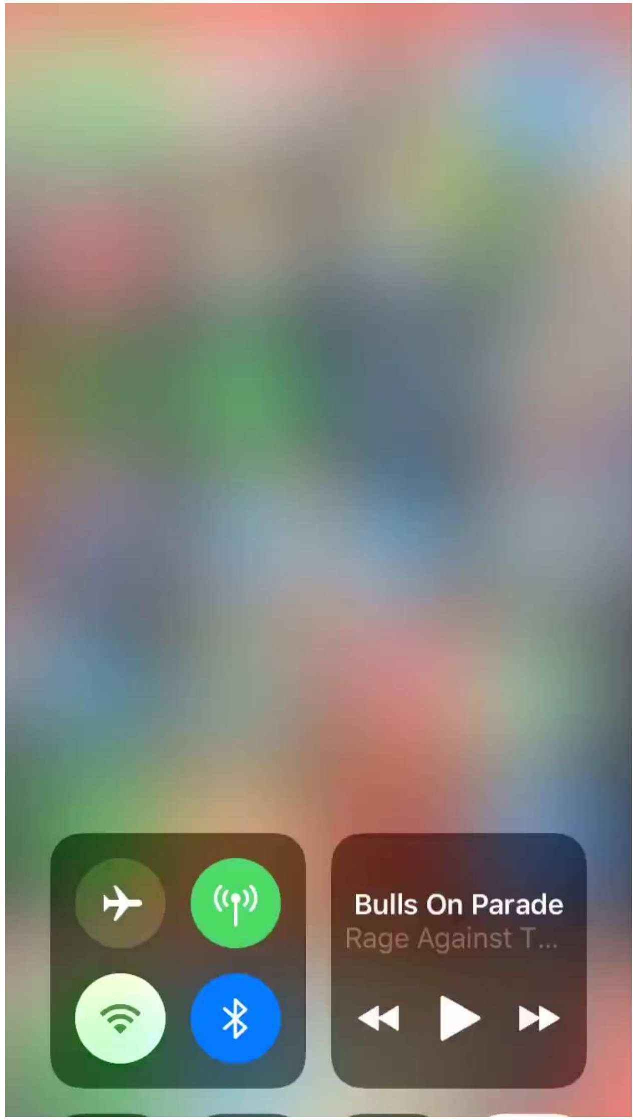
```
/* It will retrieve the next N most recent item that has been appended to the linkedlist.
To use this, you need to specify where the algorithm should start reading.
Each object retrieved from the Linkedlist contains a PageId, and a SequenceId, this is what
you need to use to generate the pointer to dictate where to start reading.
*/
module.exports.retrieveNextMostRecent = function(id, startAfterPointer, numberOfItems, callback) {
        if (!startAfterPointer ||
        !startAfterPointer.page_id ||
        !startAfterPointer.sequence_id) {
                throw 'No valid pointer has been set';
        }

        const pointer = getValidPointer(startAfterPointer);
        retrieveNElement(id, pointer.page_id, pointer.sequence_id, numberOfItems, callback);
}
```

So what do we have?

- A Linkedlist that is distributed, horizontally scalable, best ordering, dead-lock free, space/cost optimized, Linked List built on top of a key value store
- We know the limits of it (and it is configurable)
- We can most likely build Facebook, Twitter, any feed news on top of it.
    - I’ve load tested; 100 concurrent users in the same 25ms max per Linkedlist. Horizontally scale to billions of linkedlists (consistent hashing FTW)
- It would be contained in a single dependency (easy to maintain)
- Only 427 lines of code
- Most the hard things is not our problem but our PaaS provider problem: WIN
- And we never talked about technology ☺

Video



Last advices

Let the complex distributed systems thingy be owned by someone else trustworthy (hardware, replication, concensus etc…)

Stay Simple

Be a hacker

But do everything well

Use Science
(data strucrtures, TLA+ and co)

Use recovery programming and Operations readiness day 1

Have a plan, understand that scale is a journey

Be rigourous

Q&A