

## 5. Gestion des données et vie privée du citoyen urbain

- Architectures de gestion de données face au respect de la vie privée
- Gestion de la vie privée dans les réseaux sociaux mobiles
- **Privacy-by-design: gestion de données confinées (puces et capteurs)**
- Gestion de la vie privée dans les applications mobiles participatives
- Traitements de données globaux respectueux de la vie privée

Nicolas Anciaux

VILLES INTELLIGENTES : DÉFIS TECHNOLOGIQUES ET SOCIÉTAUX

Cette 3ème séquence présente les techniques de gestion de données que l'on peut embarquer dans de petits dispositifs intelligents, comme des capteurs du matériel de quantified self ou des petits objets sécurisés, disséminés dans l'environnement de l'utilisateur.

Ces techniques sont sous-jacentes aux architectures privacy by design qu'on va souhaiter implanter.

## Architectures Privacy-by-Design pour les villes

- Ce que ça n'est pas ?
  - Une architecture pour cacher des secrets

2

C'est une architecture qui va chercher à respecter la vie privée dès la conception. Première remarque, il **ne s'agit pas d'une architecture qui permet aux individus de cacher des secrets**.

## Architectures Privacy-by-Design pour les villes

- Ce que ça n'est pas ?
  - Une architecture pour cacher des secrets
- **Ce que c'est ?**
  - Permettre à l'individu de contrôler la collecte, l'usage et le partage de ses données
  - Proscrire par construction les usages secondaires « cachés » réalisés sans consentement

3

Le privacy by design a plutôt pour but, de permettre aux individus de contrôler leurs données personnelles, depuis la collecte, jusqu'à l'usage et au partage qui en sera fait. Il s'agit aussi de se prémunir par construction de tous les usages cachés ou réalisés sans le consentement de l'individu concerné.

## Architectures Privacy-by-Design pour les villes

- Ce que ça n'est pas ?
  - Une architecture pour cacher des secrets
- Ce que c'est ?
  - Permettre à l'individu de contrôler la collecte, l'usage et le partage de ses données
  - Proscrire par construction les usages secondaires « cachés » réalisés sans consentement
- **Comment faire ?**
  - Privacy-by-Design
  - Protection du cycle de vies des données de bout en bout
  - Réguler la dissémination des données depuis les sources  
... jusqu'au cloud

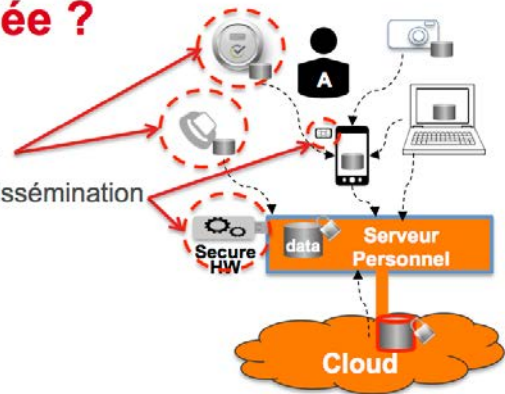
4

Comment faire ?

Il faut **réguler le cycle de vie des données de bout en bout**, depuis les sources de données jusqu'au cloud.

## Pourquoi l'évaluation confinée ?

- P-b-D: réguler le cycle de vie des données
  - De petits objets assurent une part de la collecte
  - Des composants sécurisés peuvent réguler la dissémination



5

Ces techniques aident à réguler la dissémination de données :

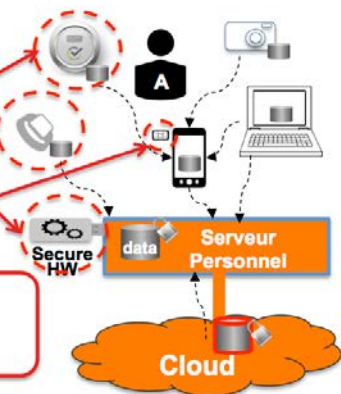
- depuis les petits objets intelligents qui vont en assurer la collecte, les capteurs,
- depuis les petits objets sécurisés qui vont permettre d'en assurer la dissémination en offrant à l'individu des garanties de non-contournement.

## Pourquoi l'évaluation confinée ?

- P-b-D: réguler le cycle de vie des données
  - De petits objets assurent une part de la collecte
  - Des composants sécurisés peuvent réguler la dissémination

Disséminer des résultats et non les données brutes  
Gestion de données embarquée pour objets intelligents

- Techniques de gestion de données pour petits objets
  - Pour ne pas surcharger les serveurs / l'infrastructure



6

Ces objets intelligents doivent être à même, de **calculer des résultats** autorisés et de **les extraire pour les transmettre à l'extérieur** plutôt, que de transmettre systématiquement toutes les données brutes collectées.

Cela passe par l'intégration de techniques de gestion de données au sein de ces dispositifs. Il y a une littérature importante sur la question. Aussi, pour d'autres raisons. Donc en particulier, parce que dans d'autres contextes, on a aussi besoin de ne pas transmettre toutes les données collectées à l'infrastructure pour ne pas surcharger les serveurs ou les réseaux de communication.

## Gestion de données : une architecture commune

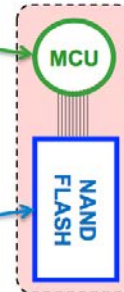
(capteurs à mémoire, token/clé USB/carte SD sécurisés, carte SIM multimédia, IoT...)

### Microcontrôleur

- Faible coût
- Faible consommation
- Protection contre les attaques physiques
  - ✓ Miniaturisation, composants superposés,
  - ✓ Maillage (signal porteur), Capteurs (lum./freq./volt.)

### Mémoire FLASH

- Faible coût
- Faible consommation
- Densité et robustesse



7

Du point de vue de la gestion de données, tous les objets intelligents partagent une même architecture qui soit petit capteur, token sécurisé, carte SIM du téléphone portable, ou carte SIM future génération à grande mémoire.

Tous disposent d'un **microcontrôleur** qui va pouvoir réaliser les traitements et d'une **mémoire de type NAND FLASH** qui assure le stockage.

Ces choix architecturaux sont liés :

- au **faible coût** et à la **faible consommation** de ces 2 composants,
- au fait que le microcontrôleur peut être protégé contre les attaques physiques dans le cas d'objets sécurisés,
- à la densité et robustesse des mémoires Flash.

## Gestion de données : une architecture commune

(capteurs à mémoire, token/clé USB/carte SD sécurisés, carte SIM multimédia, IoT...)

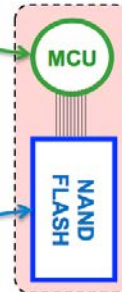
### Microcontrôleur

- Faible coût
- Faible consommation
- Protection contre les attaques physiques
  - ✓ Miniaturisation, composants superposés,
  - ✓ Maillage (signal porteur), Capteurs (lum./freq./volt.)

### Mémoire FLASH

- Faible coût
- Faible consommation
- Densité et robustesse

Très petite RAM  
(<128 KB)



### Ecritures aléatoires coûteuses

Time (ms) per I/O	Read	Seq. Write	Rand. Write
Kingston µSDHC	1.3	6.3	160
Lexar SDMI4GB-715	0.8	2.1	180
Samsung µSDHC Plus	1.3	2.9	315
SiliconPower SDHC	1.4	3.4	40

8

Cette architecture pose 2 contraintes techniques, qui ont un gros impact sur les techniques de gestion de données qu'on va être capable d'embarquer :

- le microcontrôleur, dispose de très, très peu de mémoire vive. Donc, quelques dizaines de ko dans la pratique.
- la mémoire flash, elle, offre des performances en écritures aléatoires, qui sont assez mauvaises. De l'ordre de la centaine de millisecondes, pour écrire quelques ko de données sur une carte SD de manière aléatoire.



## Gestion de données : une architecture commune

(capteurs à mémoire, token/clé USB/carte SD sécurisés, carte SIM multimédia, IoT...)

### Microcontrôleur

- Faible coût
- Faible consommation
- Protection
  - ✓ Miniaturisation
  - ✓ Maillage



Très petite RAM  
(<128 KB)

**Contraintes fortes et durables  
inhérentes aux dispositifs autonomes,  
à bas coût (capteur, carte SD),  
ou sécurisés (puce sécurisée ou "durcie")**

### Mémoire FLASH

- Faible coût
- Faible consommation
- Densité et robustesse

### Ecritures aléatoires coûteuses

Time (ms) per I/O	Read	Seq. Write	Rand. Write
Kingston µSDHC	1.3	6.3	160
Lexar SDMI4GB-715	0.8	2.1	180
Samsung µSDHC Plus	1.3	2.9	315
SiliconPower SDHC	1.4	3.4	40

9

Ces contraintes sont considérées comme inhérentes à tout dispositif autonome, à très bas coût et sécurisé. Elles sont durables.

Réaliser des traitements de données dans de tels dispositifs est une tâche difficile.

## Formulation du problème

- **Objectif** : réaliser des traitements de données (requêtes) avec peu de RAM sur de grandes quantités de données stockées en FLASH NAND

**Evaluer des requêtes  
avec une petite RAM**

*Evaluation en pipeline*



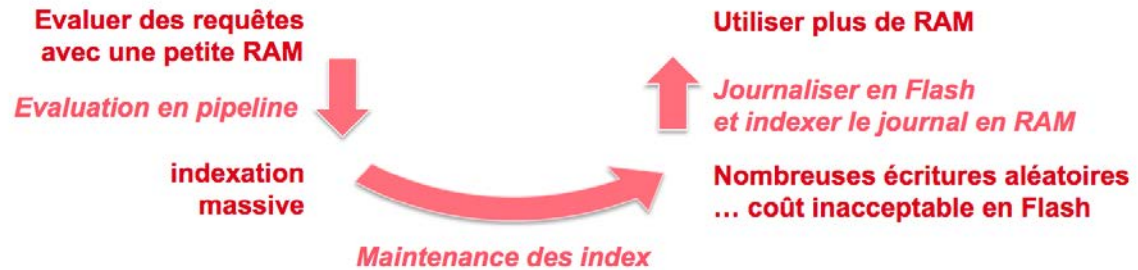
**indexation  
massive**

10

Supposons, par exemple que l'on veut calculer le résultat d'une requête avec très peu de RAM, sur de grands volumes de données stockés en mémoire flash. Que se passe-t-il ? Première chose pour évaluer une requête avec une toute petite RAM, nous devons éviter de construire des résultats intermédiaires dans cette toute petite RAM. Cela va nous conduire à consommer les données en pipeline lors du traitement. Pour réaliser un traitement en pipeline, il va falloir construire au préalable de très nombreux index sur les données.

## Formulation du problème

- **Objectif** : réaliser des traitements de données (requêtes) avec peu de RAM sur de grandes quantités de données stockées en FLASH NAND

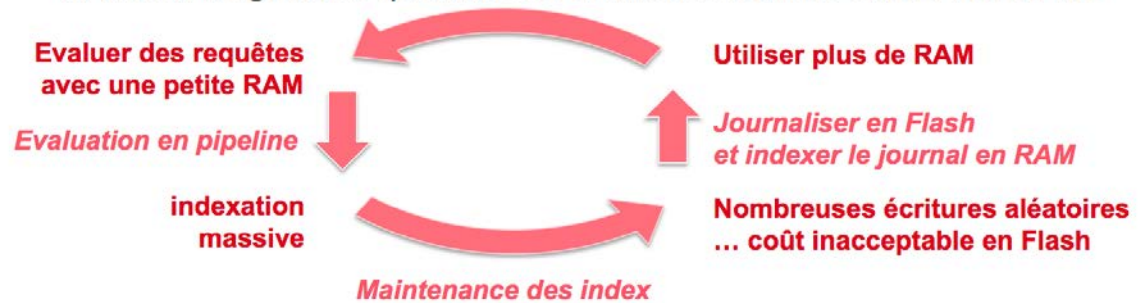


11

Mais maintenir ces index, au moment des insertions de nouvelles données, va générer de nombreuses écritures aléatoires dans la mémoire flash. Ces écritures aléatoires étant très coûteuses en flash, il faudrait pouvoir les compenser.

## Formulation du problème

- **Objectif** : réaliser des traitements de données (requêtes) avec peu de RAM sur de grandes quantités de données stockées en FLASH NAND



**Comment rompre ce cercle vicieux ?**

12

Donc, les techniques habituelles journalisent ces mises à jour et les indexent en ram.  
Mais tout ça nécessite bien plus de RAM.  
Donc un cercle vicieux s'établit car ces contraintes sont en fait extrêmement conflictuelles.

## Stratégie générale pour résoudre ce problème

### 1. Concevoir des index permettant une évaluation en pipeline

13

Les techniques de l'état de l'art se basent sur 3 principes pour résoudre ce problème. C'est-à-dire qu'il va falloir construire les index pour éviter de générer des résultats intermédiaires en RAM pendant l'exécution.

## Stratégie générale pour résoudre ce problème

1. Concevoir des index permettant une évaluation en pipeline
2. **Organiser ces index dans des structures séquentielles**  
... qui satisfont par construction les contraintes de la Flash
  - Ecrire séquentiellement dans chaque structure (et ne jamais le modifier)  
.... de manière à proscrire les écritures aléatoires par construction
  - Allouer & libérer des zones de grande taille (multiple d'un bloc de Flash)  
.... pour éviter l'émiettement de la mémoire (ramasse miettes coûteux)

14

Ensuite, il faut être capable d'organiser ces index dans des structures purement séquentielles en flash dans le but d'éviter les écritures aléatoires par construction lorsqu'on va ajouter de nouvelles données. Les index construits de manière purement séquentielle, par nature, ne passent pas à l'échelle. Donc l'accès à ces index est en général linéaire, avec le volume de données indexées.

## Stratégie générale pour résoudre ce problème

1. Concevoir des index permettant une évaluation en pipeline
2. Organiser ces index dans des structures séquentielles
3. **Passer à l'échelle par réorganisations successives**
  - Toutes les structures séquentielles sont réécrites et transformées  
... l'opération de transformation repose sur des structures séquentielles uniquement

15

Pour passer à l'échelle, il va falloir réorganiser en tâche de fonds, ces index dans de nouvelles structures séquentielles bien plus performantes. Cette réorganisation ne devra pas non plus générer d'écriture aléatoire.

## Stratégie générale pour résoudre ce problème

1. Concevoir des index permettant une évaluation en pipeline
2. Organiser ces index dans des structures séquentielles
3. Passer à l'échelle par réorganisations successives

De nombreuses techniques sont supportées aujourd'hui en embarqué. Des résultats peuvent être calculés à l'intérieur du dispositif, sans devoir externaliser toutes les données collectées.

16

En suivant ces principes, de **nombreuses techniques de gestion de données peuvent être embarquées dans des dispositifs même extrêmement contraints**. Il est ainsi possible de disséminer des résultats de requêtes sans devoir externaliser toutes les données de base.



## Conclusion

- **Nombre croissant de fonctionnalités embarquées support au P-b-D**

- Bases de données : Antelope [Sensys11], MiloDB [DAPD14], LittleD [SAC14]
- Système clé valeurs : IonDB [CCECE15]
- Recherche textuelle : Microsearch [Tan10], SSF [Anciaux15]
- Reconnaissance d'images [Yan08]
- Gestion de traces de géolocalisation [Ton15]

- **De nombreux défis restent ouverts**

- Séries temporelles, RDF, NoSQL
- Approche générale de co-conception
  - ✓ Calibrer l'architecture et les flots de données en fonction des traitements P-b-D réalisables

17

Pour conclure, il est actuellement possible d'embarquer de nombreuses techniques de gestion de données, dans des objets intelligents, même très contraints :

- Des techniques de base de données avec des prototypes comme Antelope, Milo-DB ou LittleD.
- Des systèmes de gestion de données clés valeurs, comme IonDB,
- Des moteurs de recherche textuels comme Microsearch ou SSF.
- Des processus de reconnaissance d'images,
- Gestion de traces de géolocalisation.

Bien sûr, de nombreux défis restent ouverts.

Il reste à gérer tout type de séries temporelles, des données RDF, des systèmes noSQL.

Il reste encore à concevoir une approche générale à la co-conception, c'est-à-dire qui soit capable de calibrer l'architecture, et les flots de données en fonction des traitements privacy by design qu'on va souhaiter réaliser.

## Références

- [Cavoukian11] Cavoukian, A. (2011). "Privacy by Design - The 7 Foundational Principles." <http://www.ipc.on.ca/images/Resources/7foundationalprinciples.pdf>
- [Cavoukian12] Cavoukian, A., & Jonas, J. (2012). *Privacy by design in the age of big data*. Information and Privacy Commissioner of Ontario, Canada.
- [PlugDB] <https://project.inria.fr/plugdb/>
- [Tan10] Tan, C. C., Sheng, B., Wang, H., & Li, Q. (2010). Microsearch: A search engine for embedded devices used in pervasive computing. *ACM Transactions on Embedded Computing Systems (TECS)*, 9(4).
- [Anciaux15] Anciaux, N., Lallali, S., Popa, I. S., Pucheral, P. A Scalable Search Engine for Mass Storage Smart Objects. *PVLDB* 8(9), 2015.
- [Anciaux14] Anciaux, N., Bouganim, L., Pucheral, P., Guo, Y., Le Folgoc, L. MiloDB: a Personal, Secure and Portable Database Machine. *Distributed and Parallel Databases (DAPD)*, Vol. 32(1), 2014.
- [Ton15] Ton That, D. H., Popa, I. S., Zeitouni, K. TRIFL: A Generic Trajectory Index for Flash Storage. *ACM Transactions on Spatial Algorithms and Systems* 1(2), 2015.
- [Yan08] Yan, T., Ganesan, D., and Manmatha, R. Distributed image search in camera sensor networks. In *Proc. of the 6th ACM Conference on Embedded Network Sensor Systems, SenSys'08*, pp. 155–168, 2008.
- [Douglas14] Douglas, G., & Lawrence, R. (2014, March). LittleD: a SQL database for sensor nodes and embedded applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing* (pp. 827-832). ACM.
- [CCECE15] Fazackerley, S., Huang, E., Douglas, G., Kudlac, R., & Lawrence, R. (2015, May). Key-value store implementations for Arduino microcontrollers. In *Electrical and Computer Engineering (CCECE), 2015 IEEE 28th Canadian Conference on* (pp. 158-164). IEEE. IonDB, Link: [https://www.reddit.com/r/arduino/comments/34uu8r/iondb\\_is\\_a\\_key\\_value\\_store\\_for\\_the\\_arduino/](https://www.reddit.com/r/arduino/comments/34uu8r/iondb_is_a_key_value_store_for_the_arduino/)