Microsoft

PostgreSQL Internals and Architecture

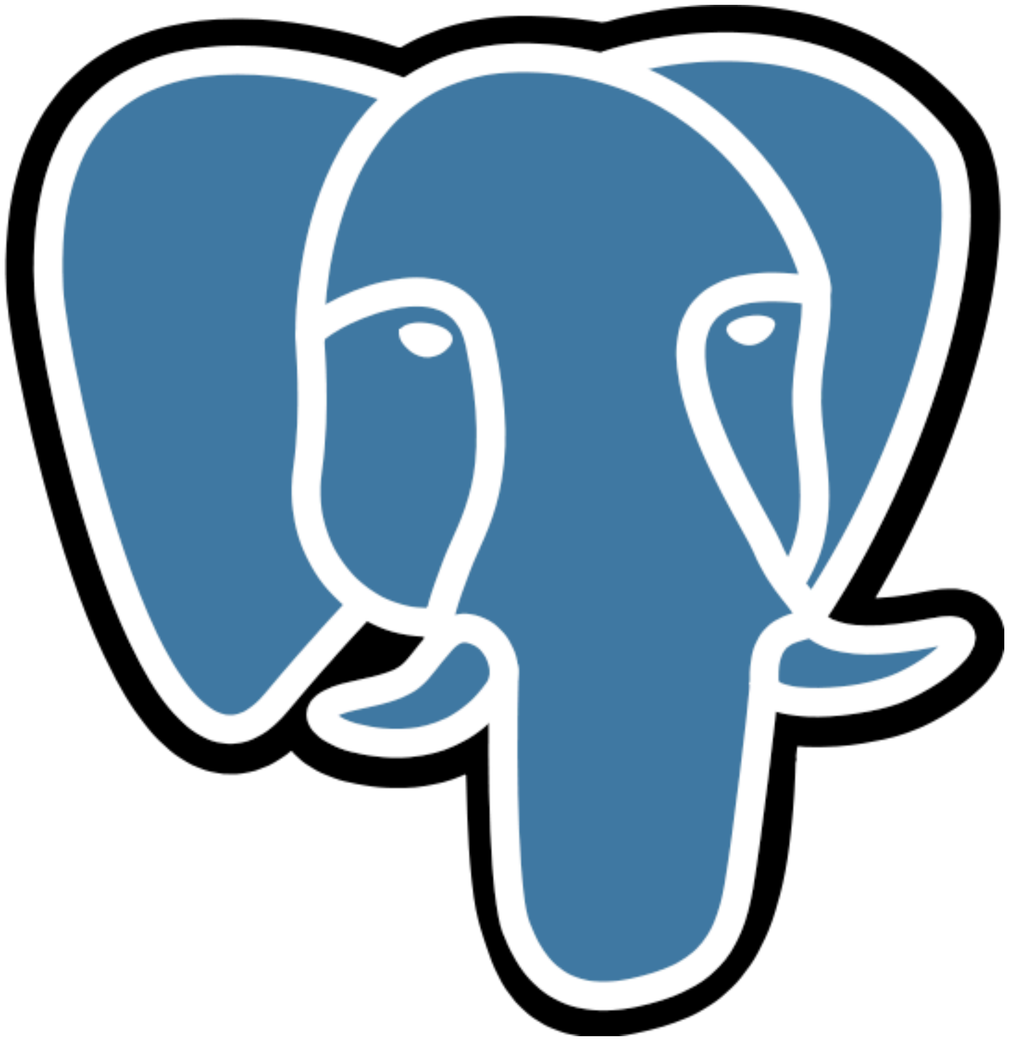
Daniel Gustafsson

Daniel Gustafsson

Hacking Postgres at Microsoft since April 2023

Based in Sweden

Timezone nomad..



Contributor since 2014

Committer (core, web)

Stockholm PUG

Nordic PGDay

PGConf EU

FOSDEM PGDay

PostgreSQL Development Conference

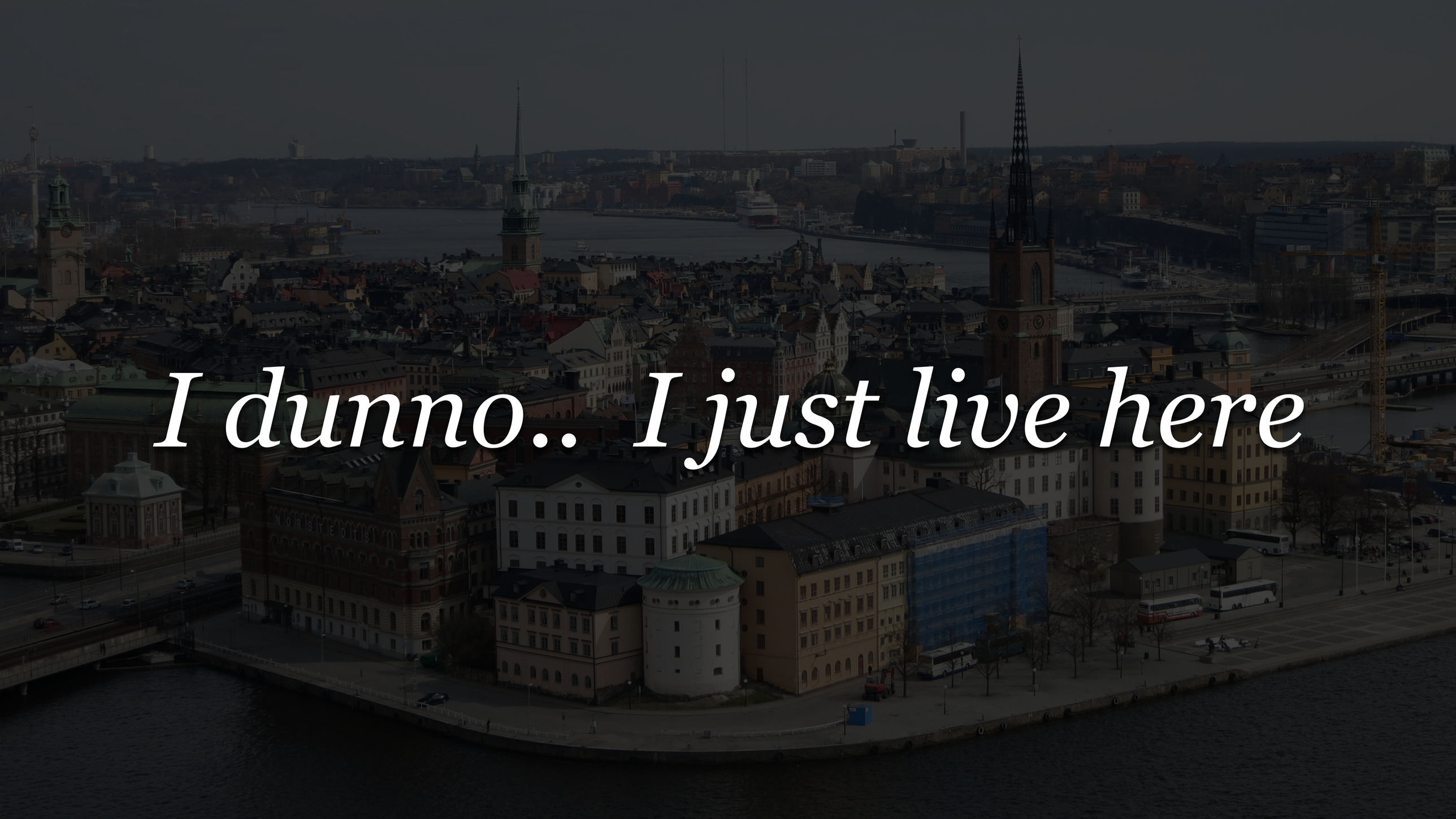
Agenda

Brief overview of history and project

An overview of components

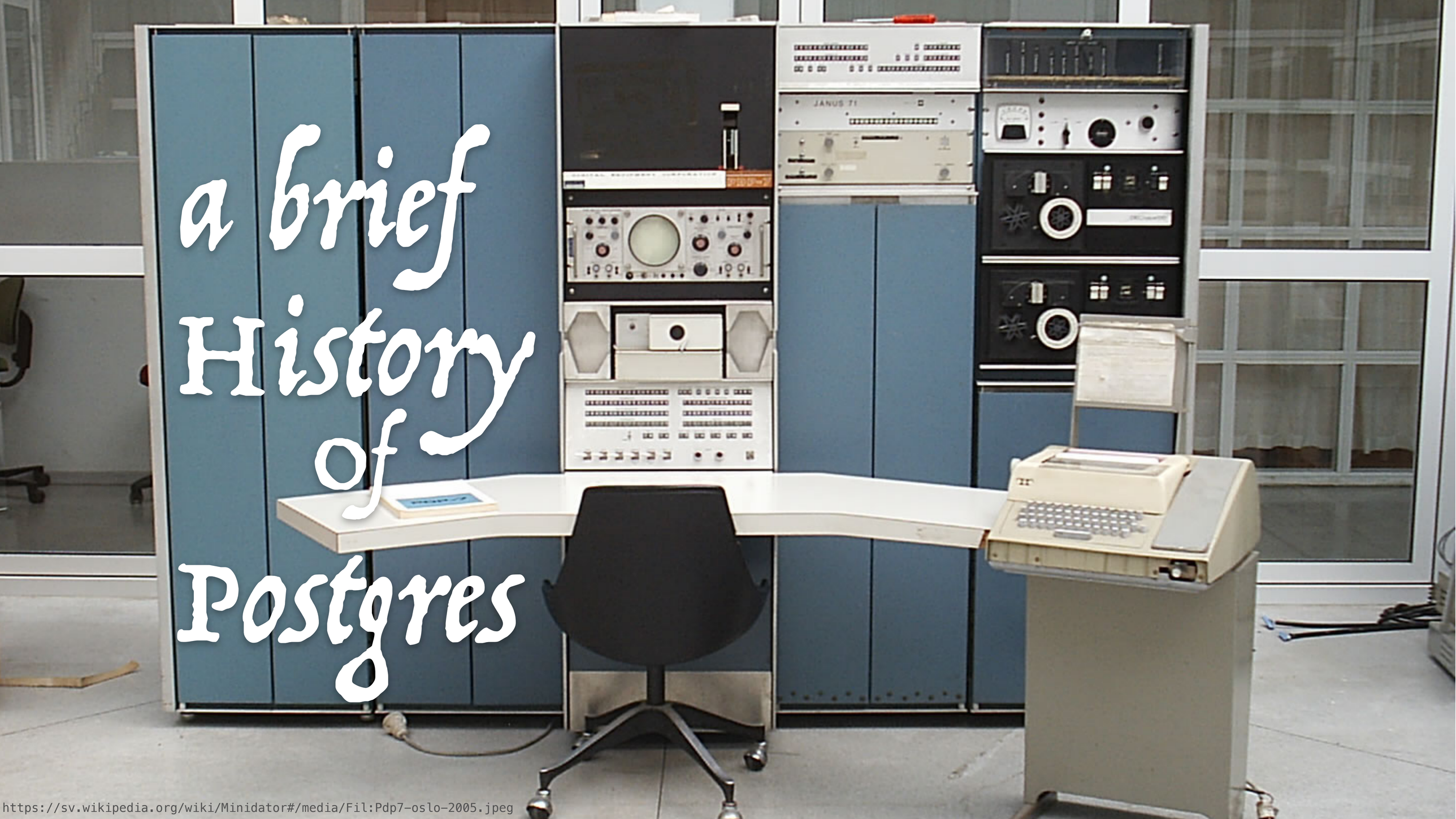
Selected deep dives

This was hard..

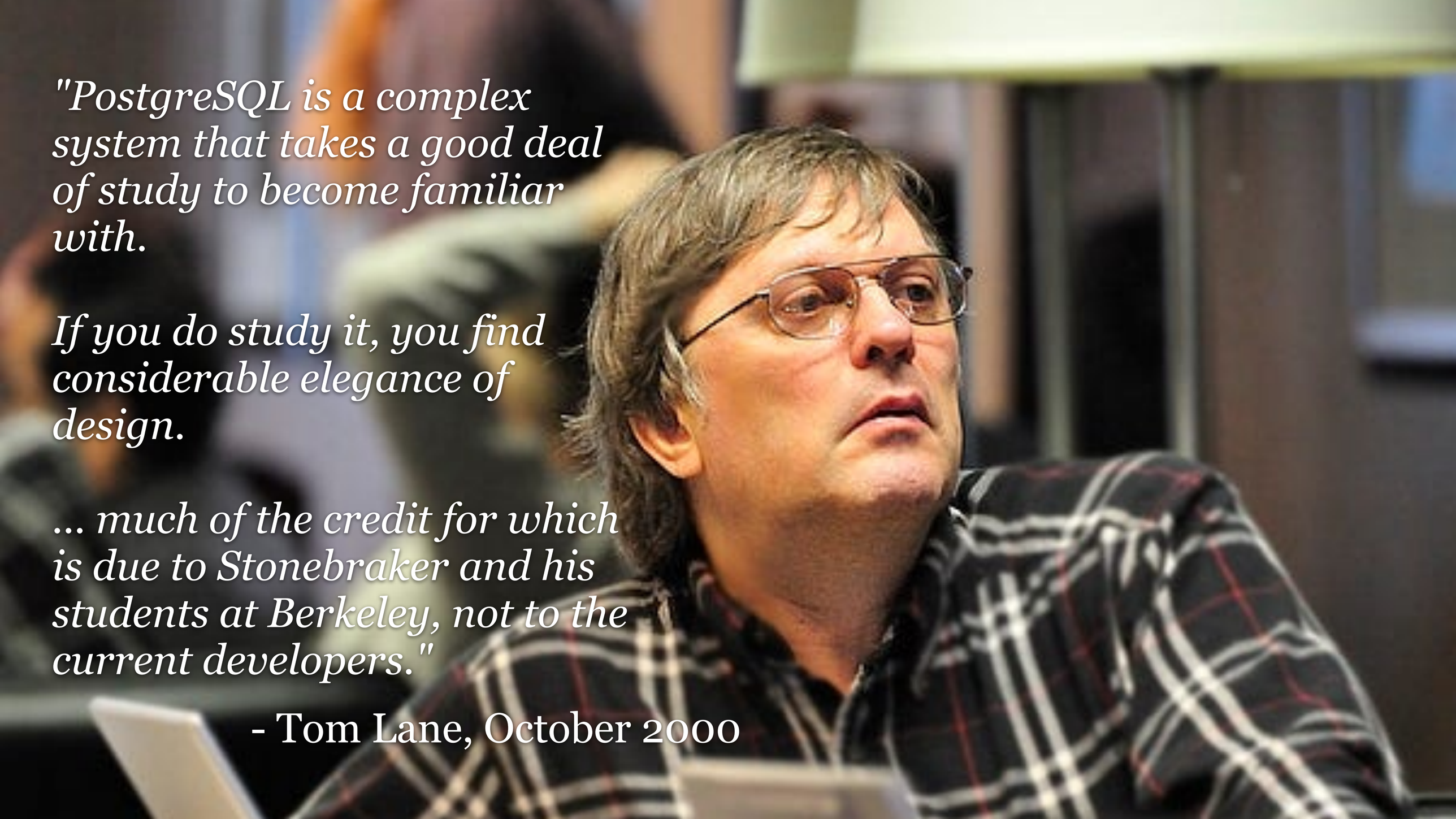


I dunno.. I just live here


WARNING
DANGER
DO NOT PULL
HANDLE
RANGER
CENTURION



https://sv.wikipedia.org/wiki/Minidator#/media/Fil:Pdp7-oslo-2005.jpeg


"PostgreSQL is a complex system that takes a good deal of study to become familiar with.
If you do study it, you find considerable elegance of design.
... much of the credit for which is due to Stonebraker and his students at Berkeley, not to the current developers."
- Tom Lane, October 2000

Origins

Research project from the lab of Mike Stonebraker at Berkeley

Initially written in Lisp

SQL didn't come until later, initially used the Quel language

Origins of Postgres of Today

Initial C revision in 1989

OSS repository dates to 1996

The PostgreSQL project was founded at the time

Development

Written in C, testharness in Perl

~25 committers, 200+ contributors

Not using Github

Mailinglists and custom apps

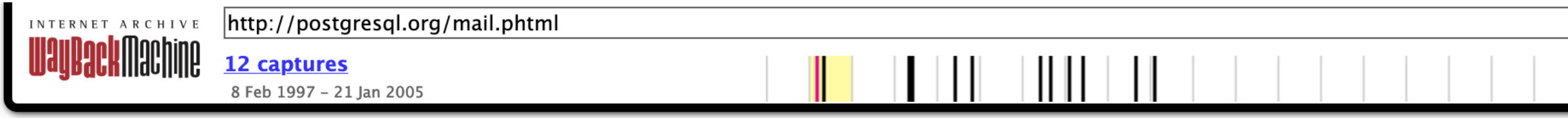
Hackers?

pgsql-hackers@ mailinglist

Nearly every commit can be traced to a -hackers discussion

Unrivalled archive of development

Hackers!



PostgreSQL Mailing Lists

General Users/Questions

> This list is designed as a forum geared towards installation and usage problems related to PostgreSQL, as well as a place to discuss gener
>
> To subscribe, send a message of **subscribe** to: questions-request@postgreSQL.org

Hackers/Development Discussions

> This list is where bugs reports and server improvements are discussed, providing the developers with an area seperate from the general q
> join in and follow the discussions.
>
> To subscribe, send a message of **subscribe** to: hackers-request@postgreSQL.org

All the mailing lists are archived, with archives available at: ftp://ftp.postgreSQL.org/pub/majordomo

Building from Source

`git.postgresql.org`

Meson, Autoconf, (MSVC)

Flex, Bison, Readline, IPC::Run

Linux, macOS, *BSD, Solaris, AIX
Windows

Design Principles

No roadmap

No design document

Developers scratching their itch

Extremely independent and autonomous

THE DESIGN OF POSTGRES

Michael Stonebraker and Lawrence A Rowe

Department of Electrical Engineering
and Computer Sciences
University of California
Berkeley, CA 94720

Abstract

This paper presents the preliminary design of a new database management system, called POSTGRES, that is the successor to the INGRES relational database system The main design goals of the new system are to

1) provide better support for complex objects,

2) provide user extendibility for data types, operators and access methods,

3) provide facilities for active databases (i e , alerters and triggers) and inferencing including forward- and backward-chaining,

4) simplify the DBMS code for crash recovery,

5) produce a design that can take advantage of optical disks, workstations composed of multiple tightly-coupled processors, and custom designed VLSI chips, and

6) make as few changes as possible (preferably none) to the relational model

THE DESIGN OF POSTGRES

Michael Stonebraker and Lawrence A Rowe

Department of Electrical Engineering
and Computer Sciences
University of California
Berkeley, CA 94720

Abstract

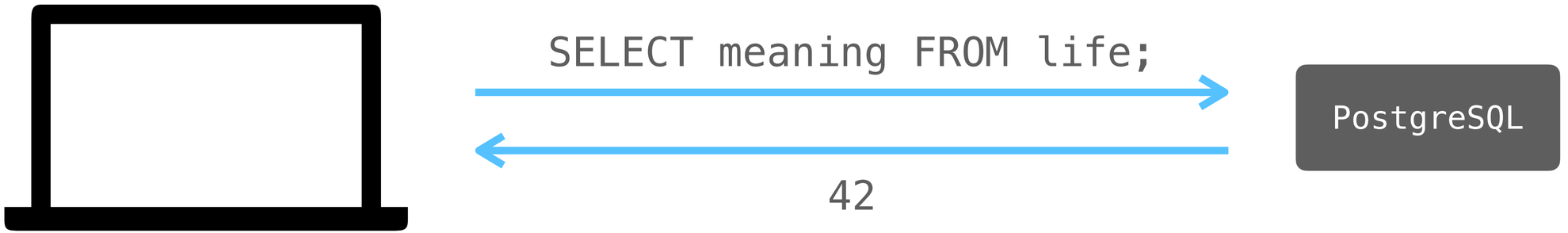
This paper presents the preliminary design of a new database management system, called POSTGRES, that is the successor to the INGRES relational database system The main design goals of the new system are to

1) provide better support for complex objects,
2) provide user extendibility for data types, operators and access methods,
3) provide facilities for active databases (i e , alerters and triggers) and inferencing including forward- and backward-chaining,
4) simplify the DBMS code for crash recovery,
5) produce a design that can take advantage of optical disks, workstations composed of multiple tightly-coupled processors, and custom designed VLSI chips, and
6) make as few changes as possible (preferably none) to the relational model


https://flickr.com/photos/febird/8965404736/

SELECT meaning FROM life;

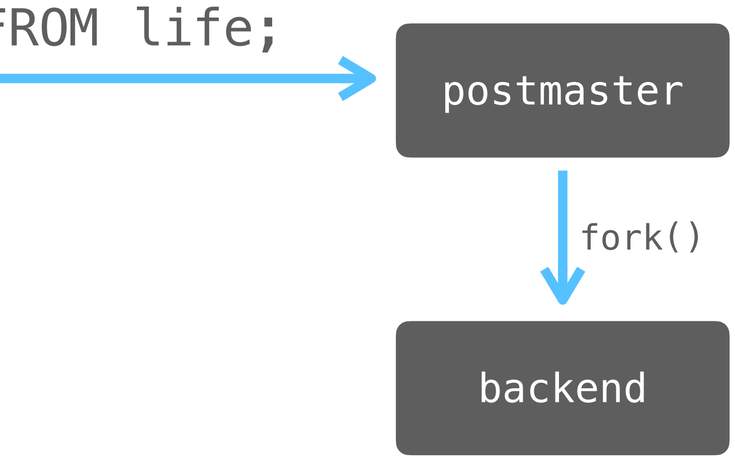
PostgreSQL


SELECT meaning FROM life;
PostgreSQL
42

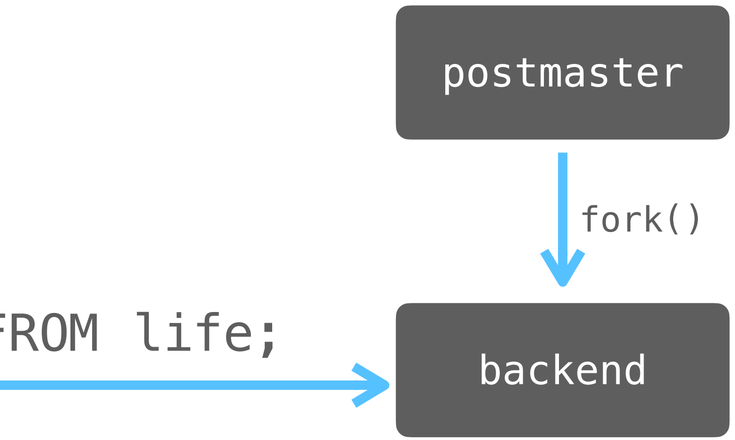
SELECT meaning FROM life;

42

FROM life;


FROM life;
postmaster
fork()
backend

PostgreSQL is using processes, not threads

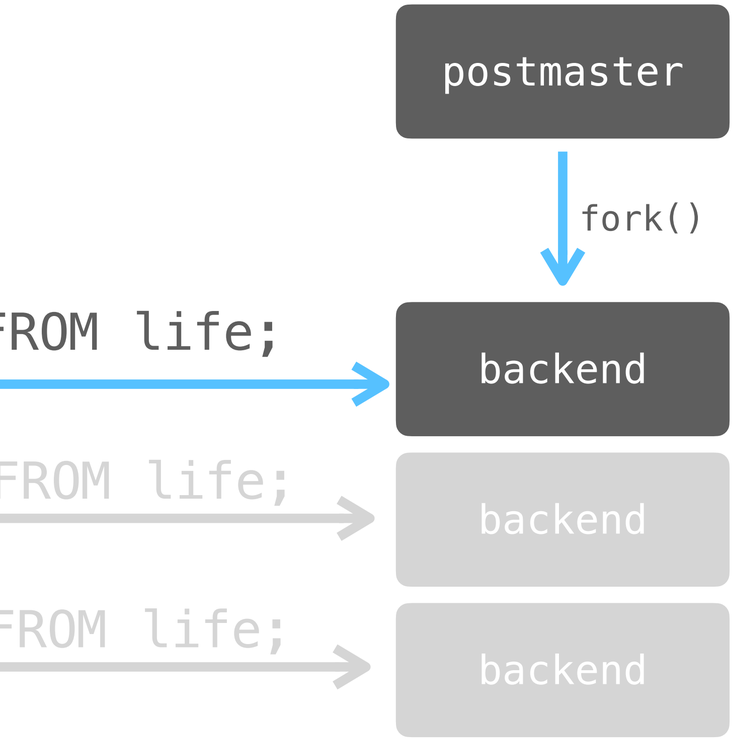


Currently, POSTGRES runs as one process for each active user. This was done as an expedient to get a system operational as quickly as possible. We plan on converting POSTGRES to use lightweight processes available in the operating systems we are using.

Stonebraker, M. et. al, "The implementation of POSTGRES",
IEEE Transactions on Knowledge and Data Engineering,
issue 1, March 1990

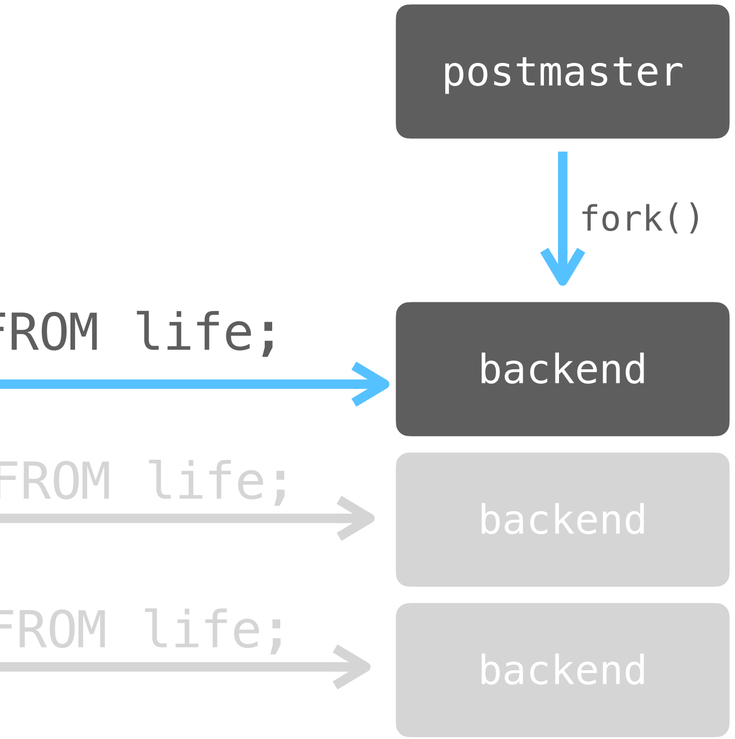
PostgreSQL is using processes, not threads

Each connection runs in a separate backend process



PostgreSQL is using processes, not threads

Each connection runs in a separate backend process



Controlled by `max_connections` GUC, by default set to 100

GUC?

Grand Unified Configuration

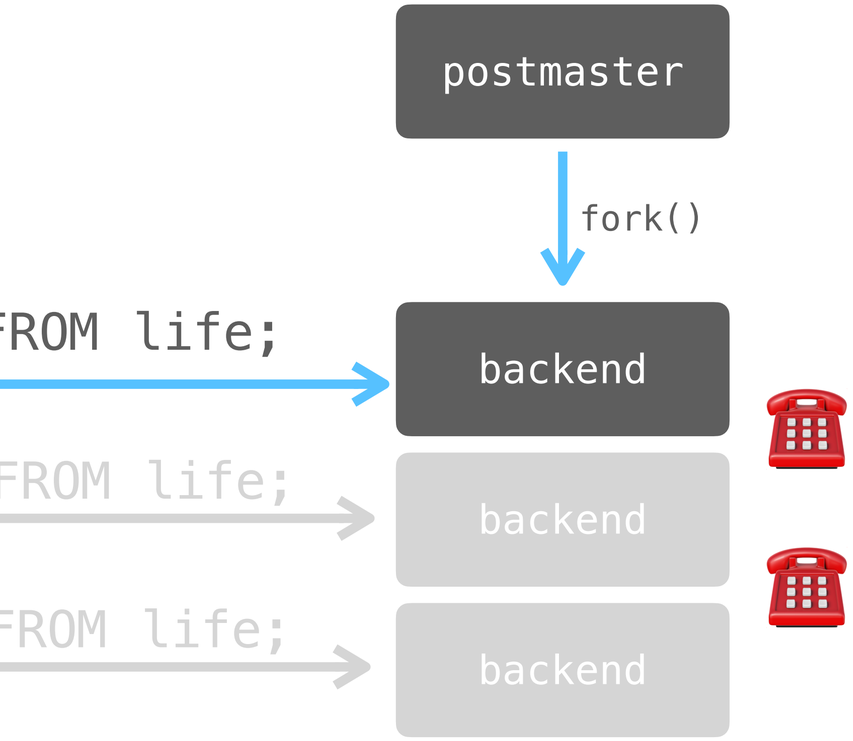
Typed configuration variables

All user-exposed configuration are implemented as GUCs

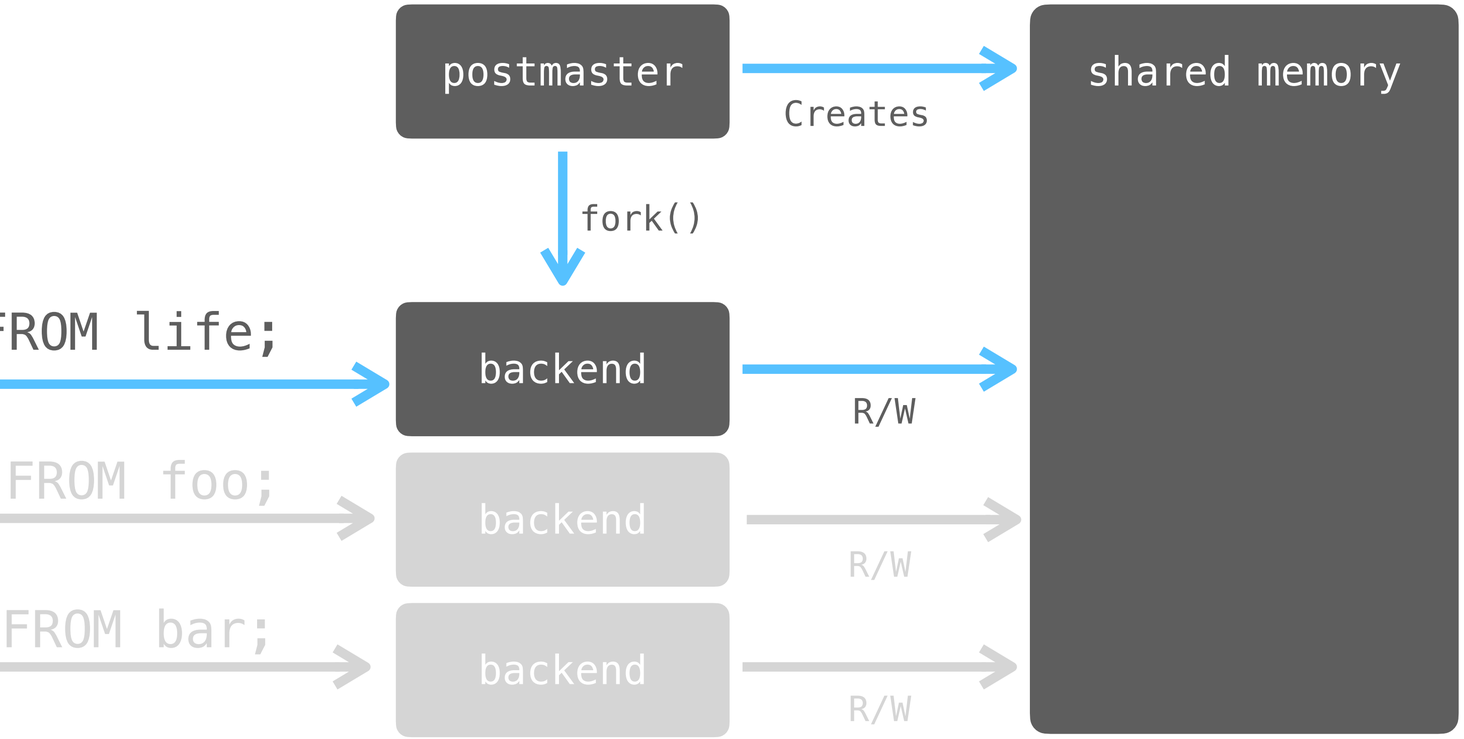
PostgreSQL is using processes, not threads

Each connection runs in a separate backend process

Processes need to communicate

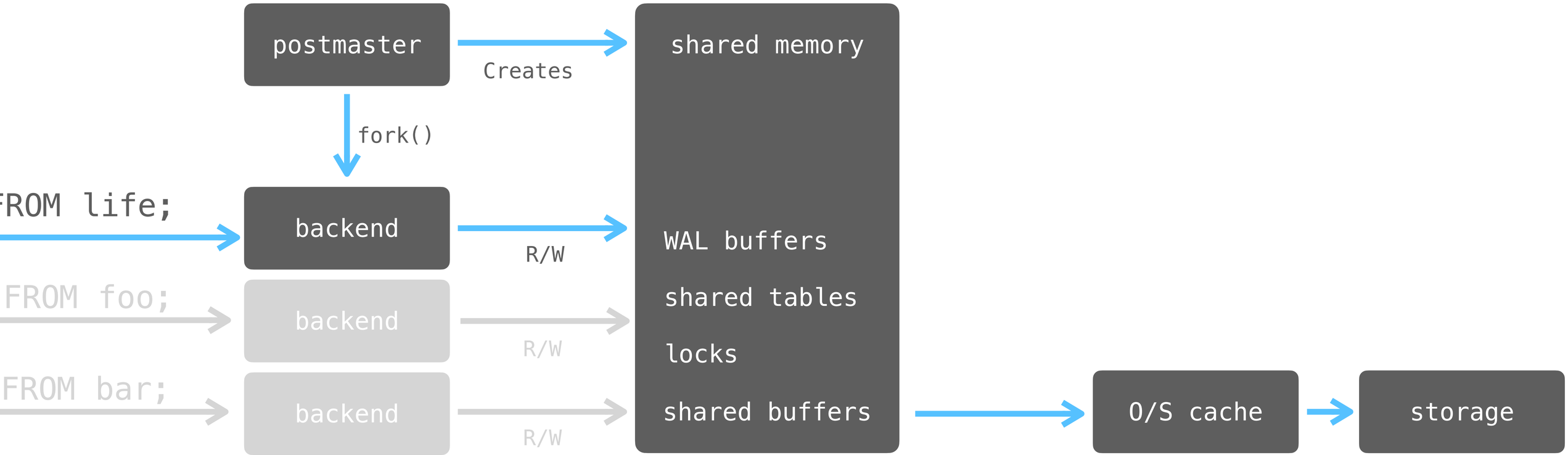


Postmaster creates shared memory



Postmaster creates shared memory

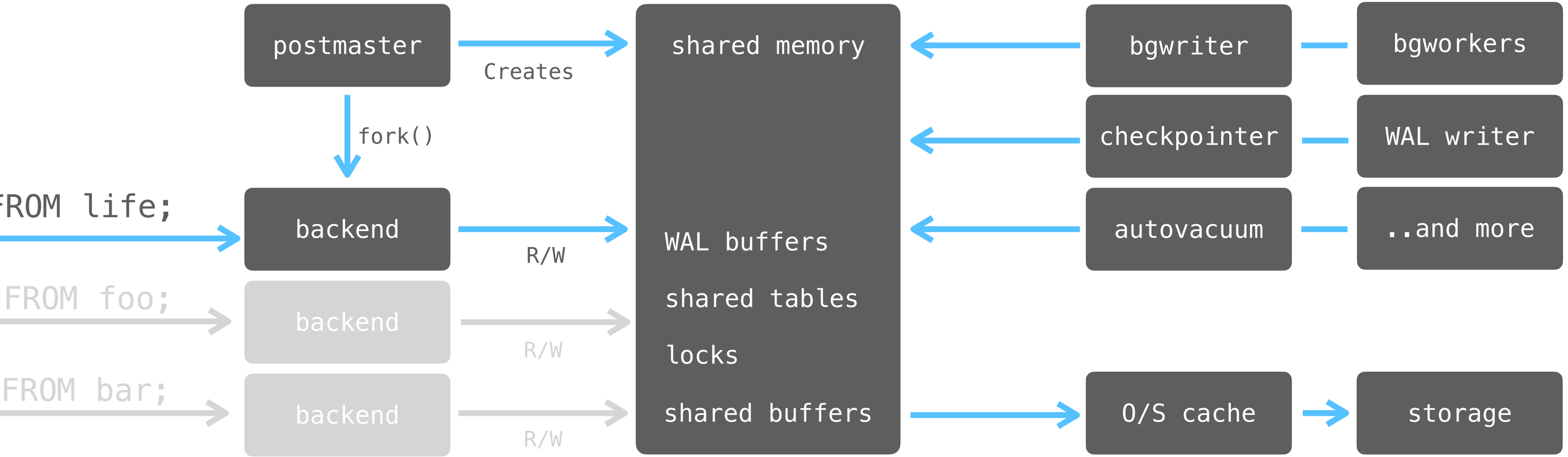
Contains structures shared between backends

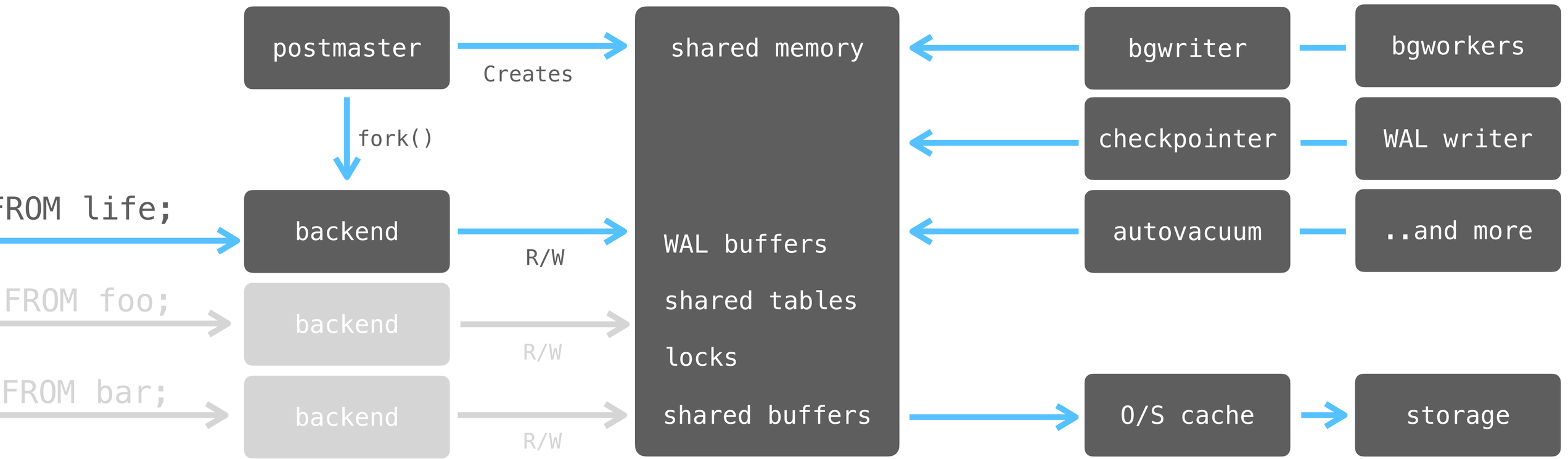


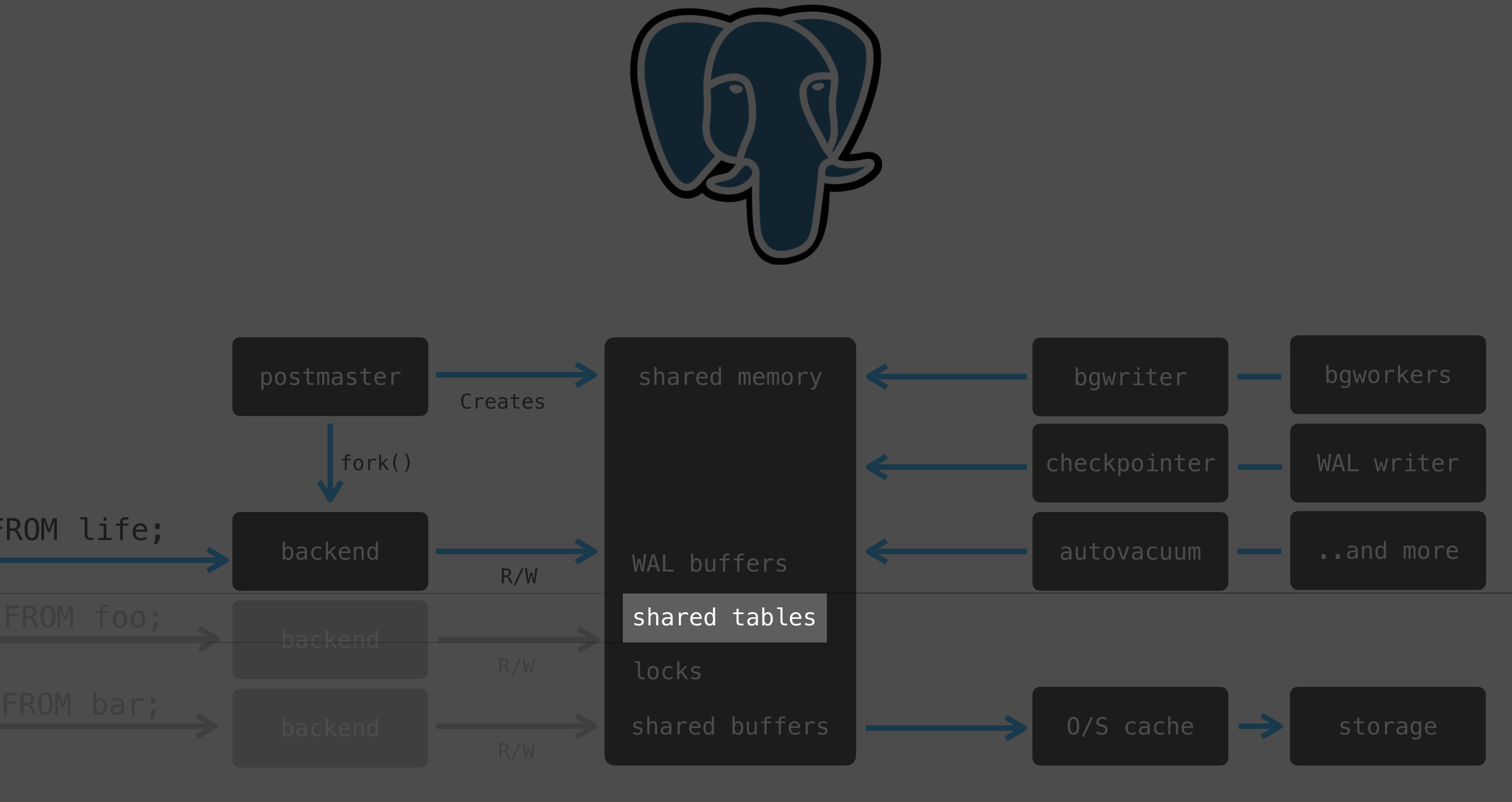
Postmaster creates shared memory

Contains structures shared between backends

Internal processes also use shared memory




postmaster
Creates
shared memory
bgwriter
bgworkers
fork()
checkpointer
WAL writer
FROM life;
backend
R/W
WAL buffers
autovacuum
..and more
FROM foo;
backend
R/W
shared tables
locks
FROM bar;
backend
R/W
shared buffers
O/S cache
storage


postmaster
Creates
shared memory
bgwriter
bgworkers
fork()
checkpointer
WAL writer
FROM life;
backend
R/W
WAL buffers
autovacuum
..and more
FROM foo;
backend
R/W
shared tables
locks
FROM bar;
backend
R/W
shared buffers
O/S cache
storage

Shared Tables

System catalog

Drives extensibility

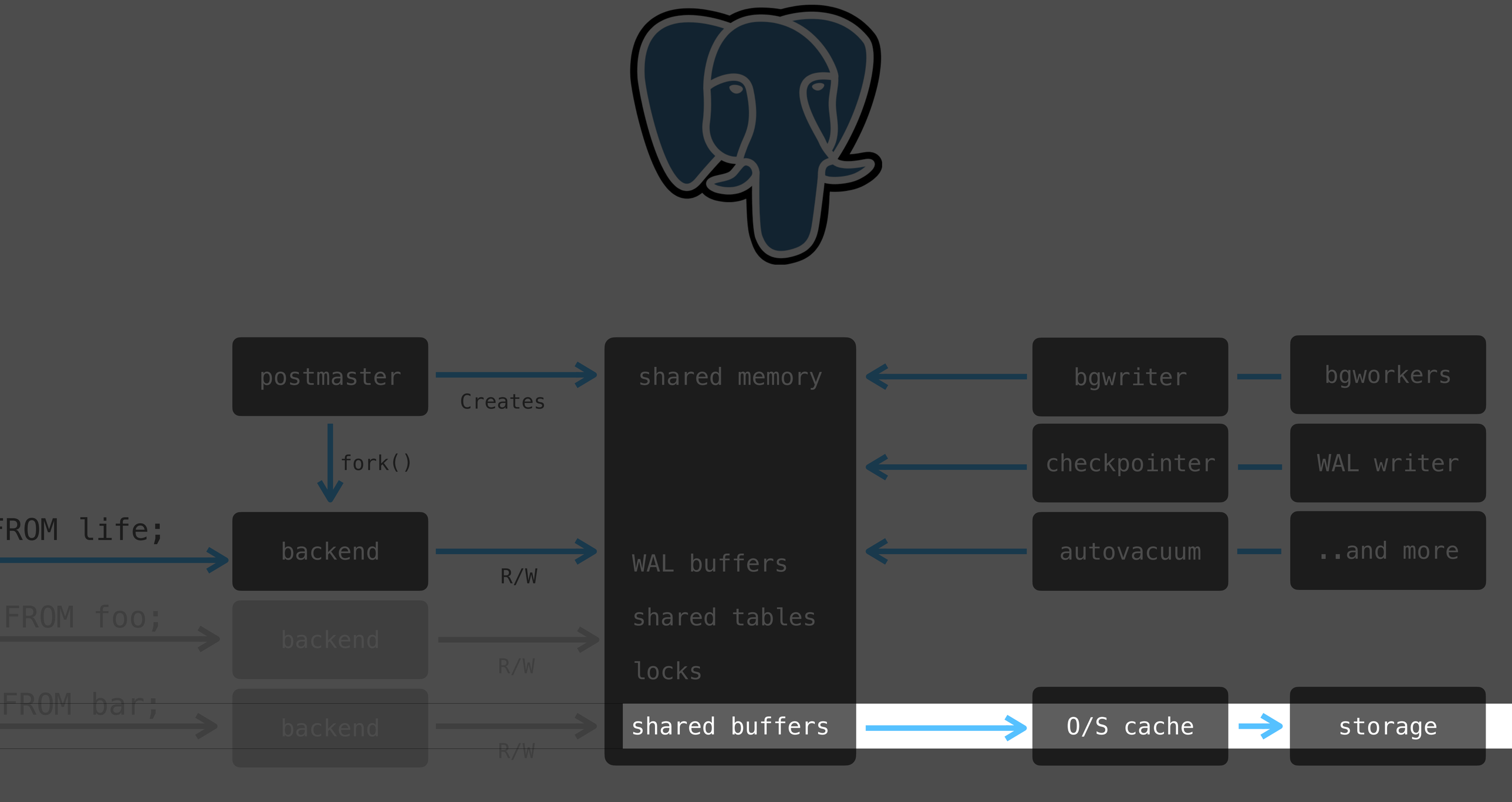
Types, Operators, Functions, Extensions, Tables, Rows, Attributes.. etc

Shared Tables

Postgres has very little hardcoded knowledge

Catalog lookups for everything

Much more used in Postgres than other systems


postmaster
Creates
shared memory
bgwriter
bgworkers
fork()
checkpointer
WAL writer
FROM life;
backend
R/W
WAL buffers
autovacuum
..and more
FROM foo;
backend
R/W
shared tables
FROM bar;
backend
R/W
locks
shared buffers
O/S cache
storage

Shared Buffers

Disk is slow, memory is fast

Doublebuffering with O/S cache

Buffer manager looks for page in shared buffers first

If found, a cache hit, it is pinned

Shared Buffers

In case of cache miss, the page is read from disk into shared buffers

Cache eviction by clock sweep

Dirty pages are written out before eviction

WAL

Disk and RAM are inconsistent

A log entry is written sequentially instead of flushing every page on changes

Write-Ahead Log

WAL

Replay to restore consistency

Where to start replaying?

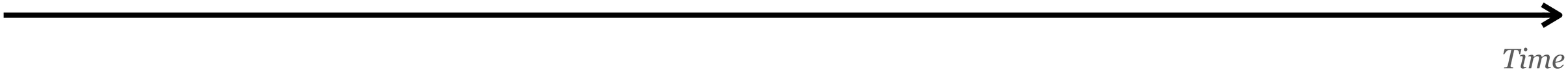
Need a safe checkpoint

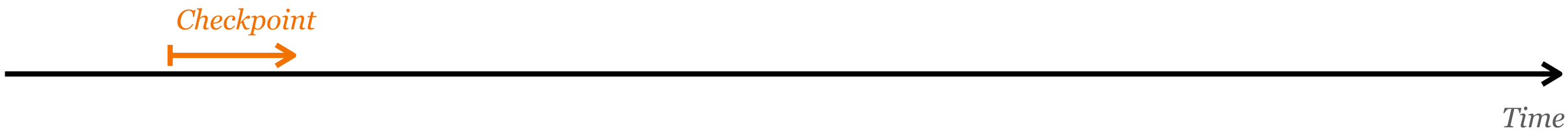
Stop the world and flush dirty buffers?

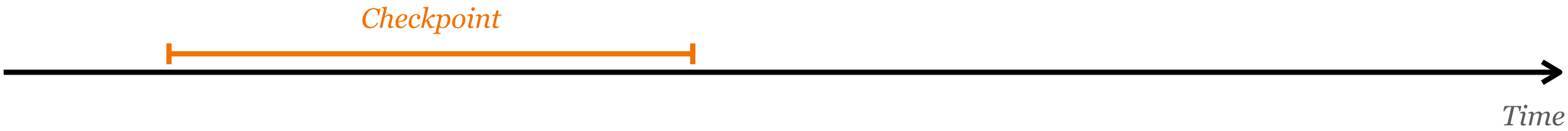
Checkpointer

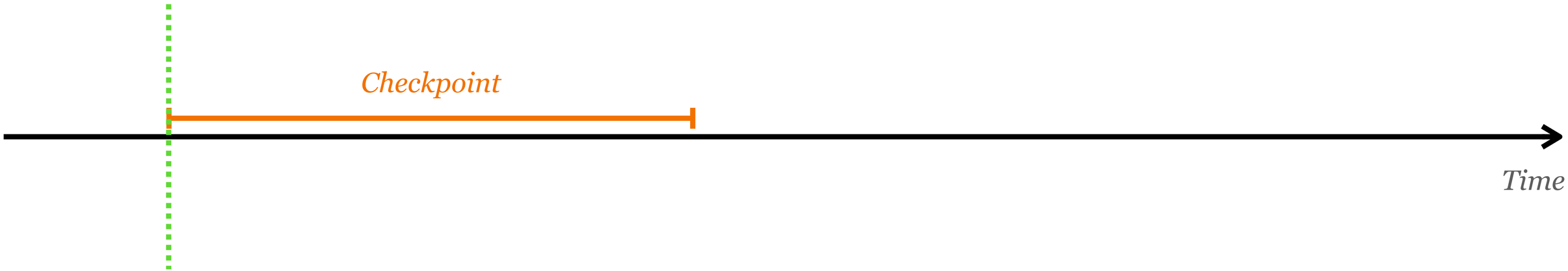
System process which over a period of time writes out dirty buffers

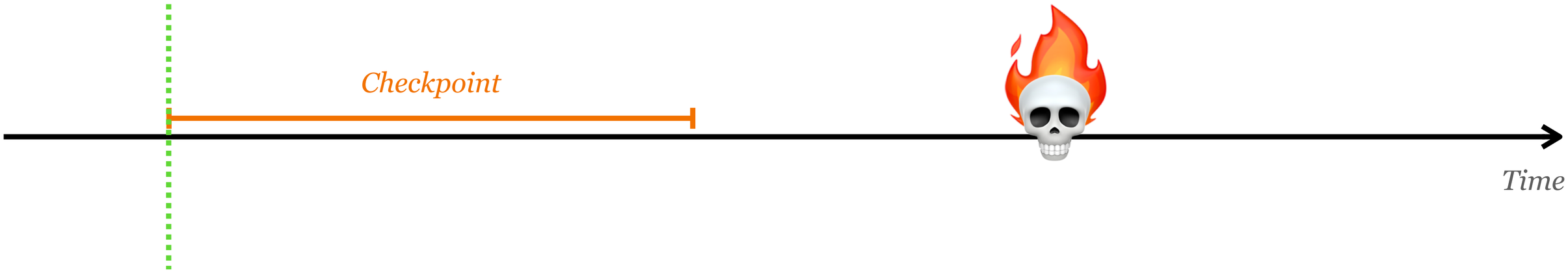
Checkpoint complete when buffers dirty at the start of checkpointing are written

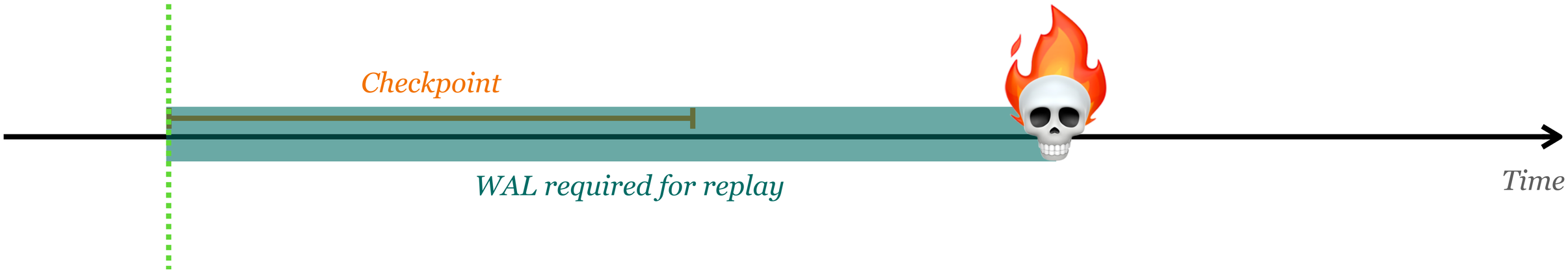

Time


Checkpoint
Time


Checkpoint
Time


Checkpoint
Time


Checkpoint
Time


Checkpoint
WAL required for replay
Time

Data on disk

Relations are stored in forks

A fork is a file with a segment limit

When limit is reached, a new segment is created

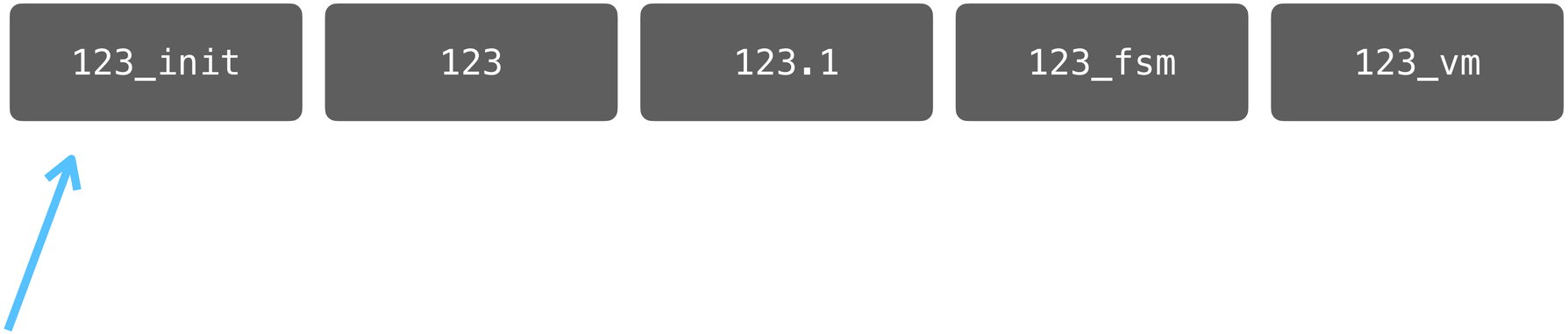
Different types of forks

Data on disk

Changing on-disk format is avoided

A change breaks `pg_upgrade` file copy/hardlinking mode

Last big change was 8.2 ➠ 8.3



Init fork

Only for UNLOGGED relations

Init fork

Only for UNLOGGED relations

Main fork

The first 1Gb segment of a relation

Init fork

Only for UNLOGGED relations

Main fork

The first 1Gb segment of a relation

Subsequent fork(s)

The next 1Gb segments of a relation, numbered 1..*n*

Init fork

Only for UNLOGGED relations

Main fork

The first 1Gb segment of a relation

Free Space Map

Stores the amount of free space on pages as an aggregated tree

Subsequent fork(s)

The next 1Gb segments of a relation, numbered 1..*n*

Init fork

Only for UNLOGGED relations

Main fork

The first 1Gb segment of a relation

Visibility Map

Tracks which pages contains tuples visible to all transactions

Free Space Map

Stores the amount of free space on pages as an aggregated tree

Subsequent fork(s)

The next 1Gb segments of a relation, numbered 1..*n*

Init fork

Only for UNLOGGED relations

Main fork

The first 1Gb segment of a relation

Visibility Map

Tracks which pages contains tuples visible to all transactions

Free Space Map

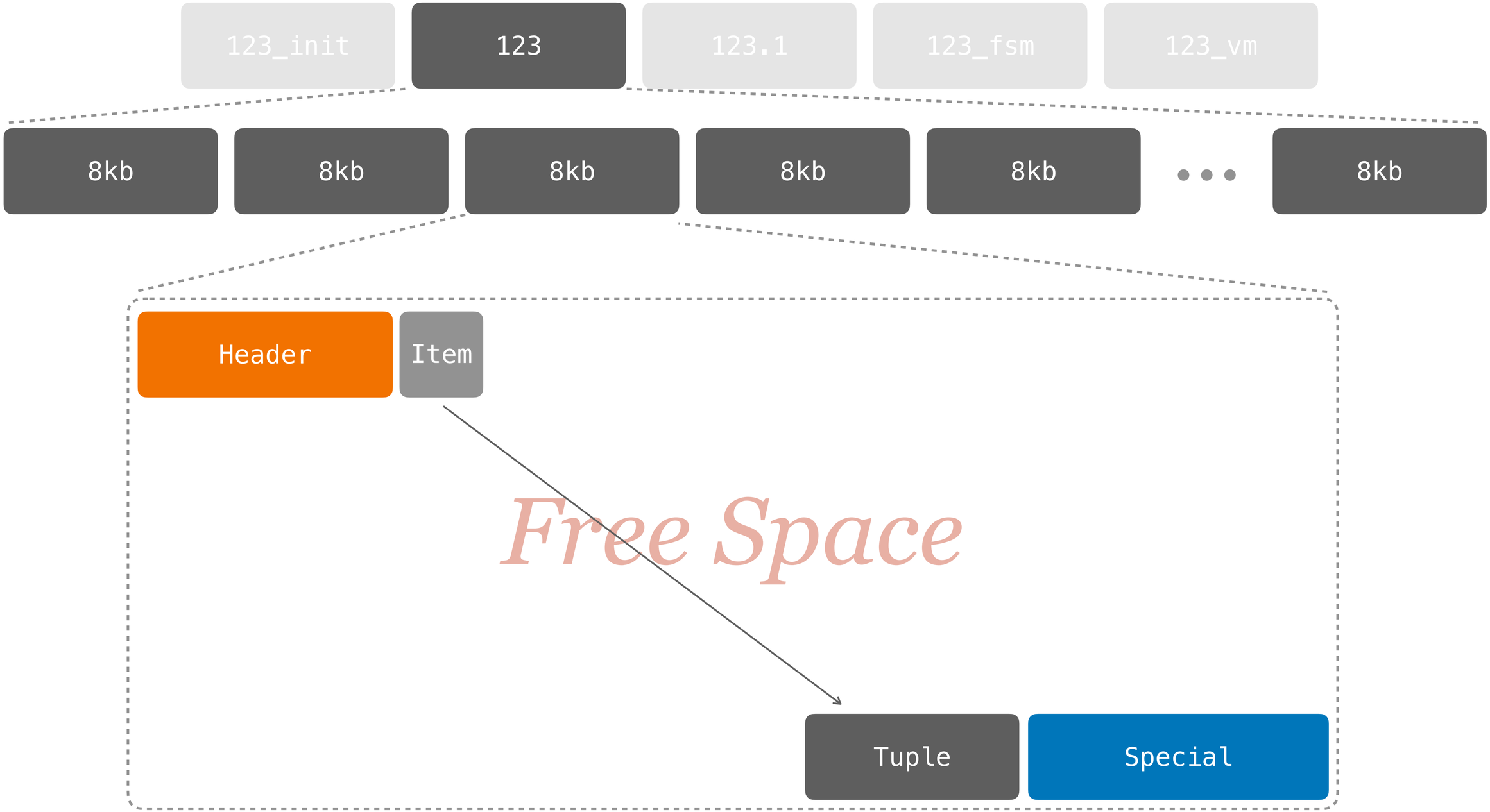
Stores the amount of free space on pages as an aggregated tree

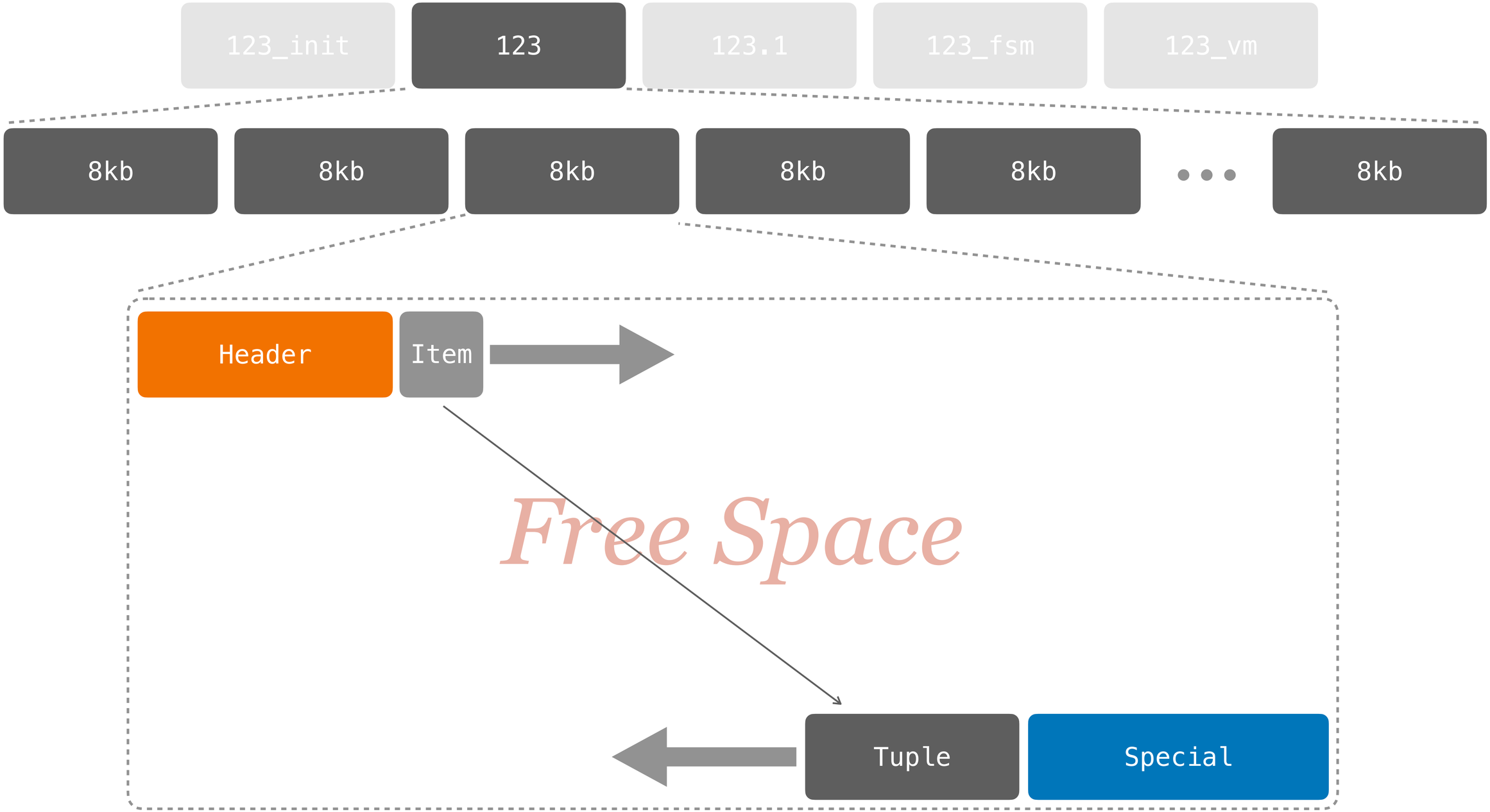
Subsequent fork(s)

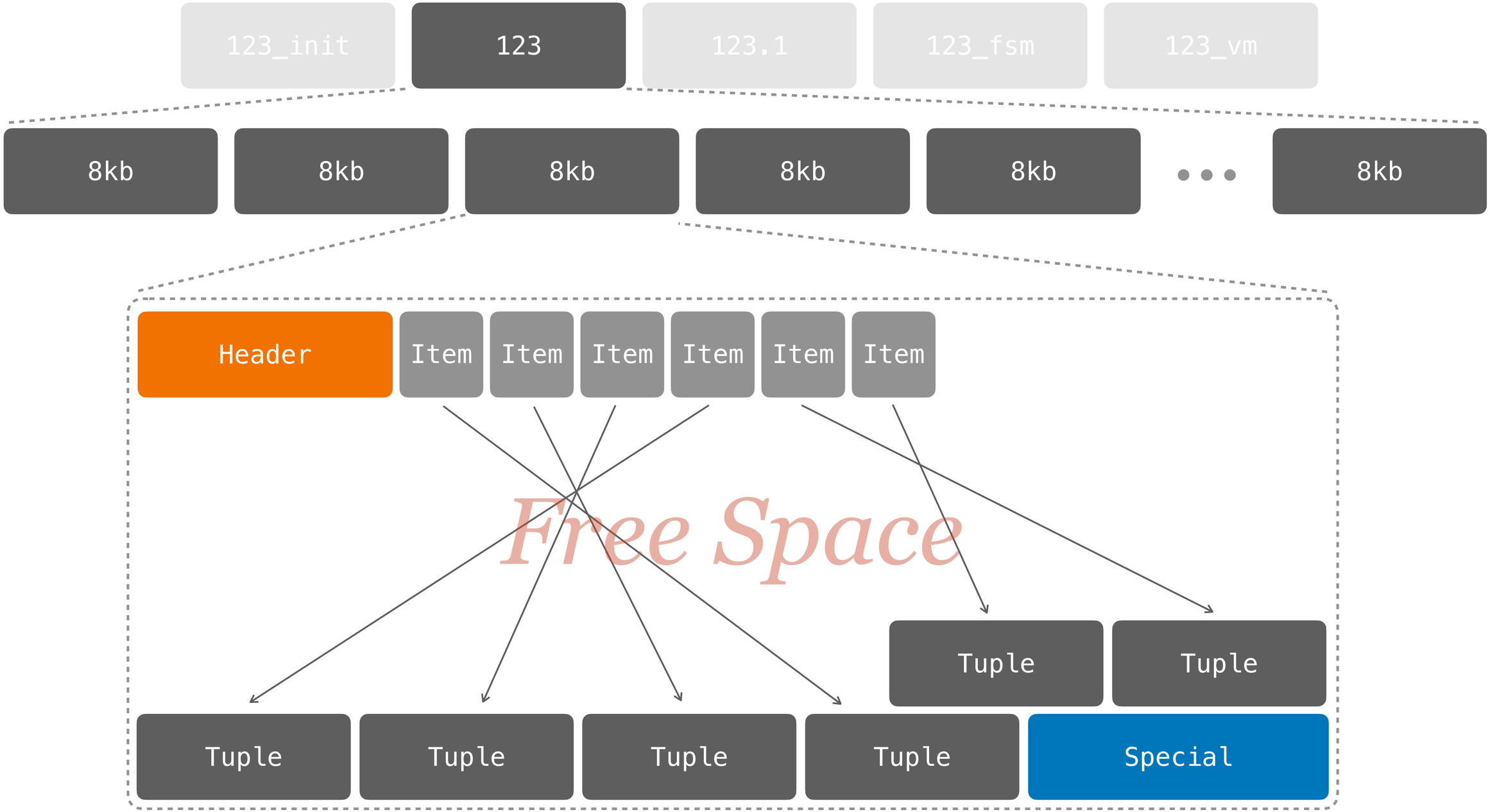
The next 1Gb segments of a relation, numbered 1..*n*

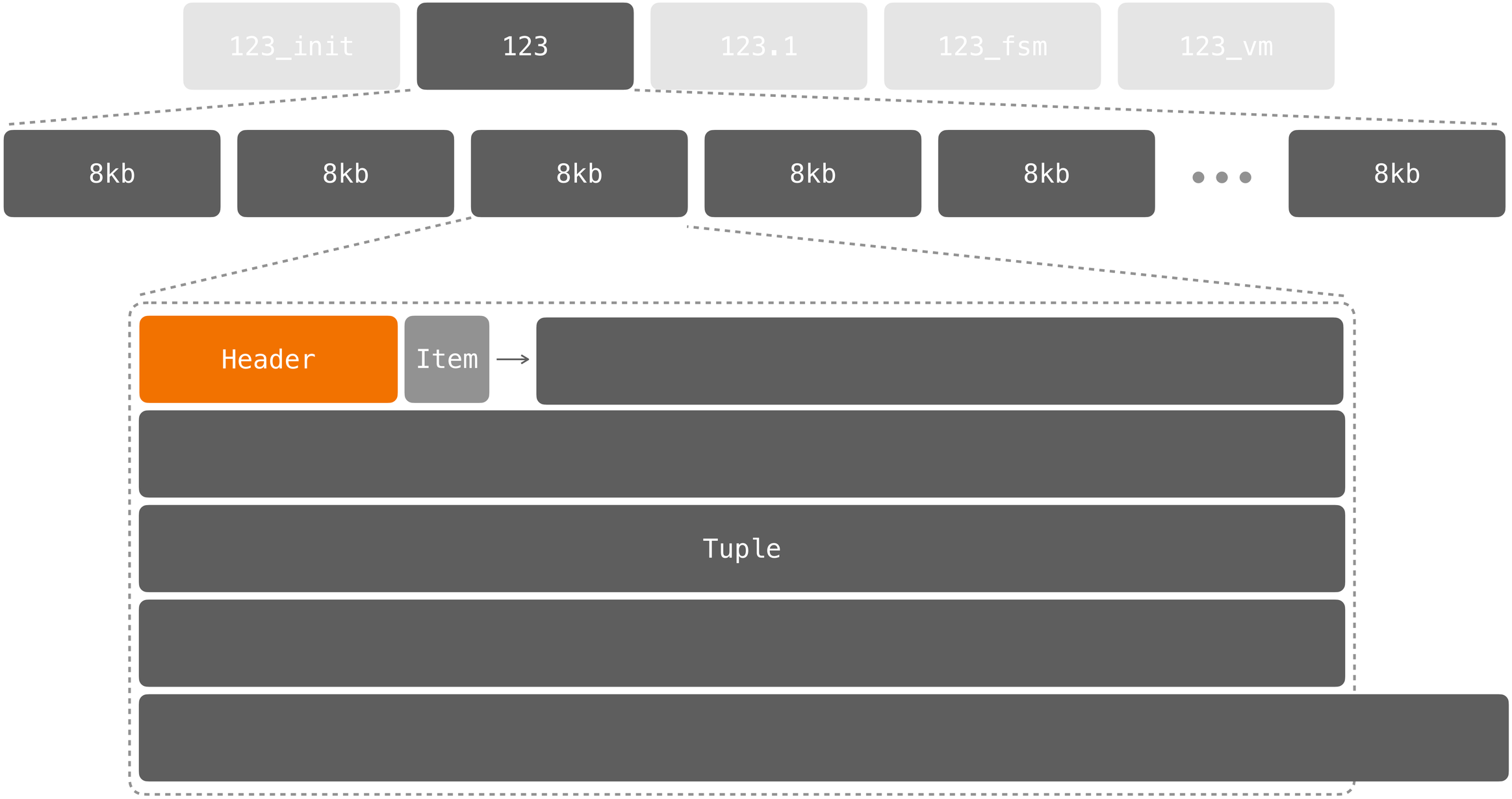

123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb


123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb
Header
Free Space
Special


123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb
Header
Item
Free Space
Tuple
Special


123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb
Header
Item
Free Space
Tuple
Special


123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb
Header
Item
Item
Item
Item
Item
Item
Free Space
Tuple
Tuple
Tuple
Tuple
Tuple
Tuple
Special


123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb
Header
Item
Tuple


123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb
Header
Item

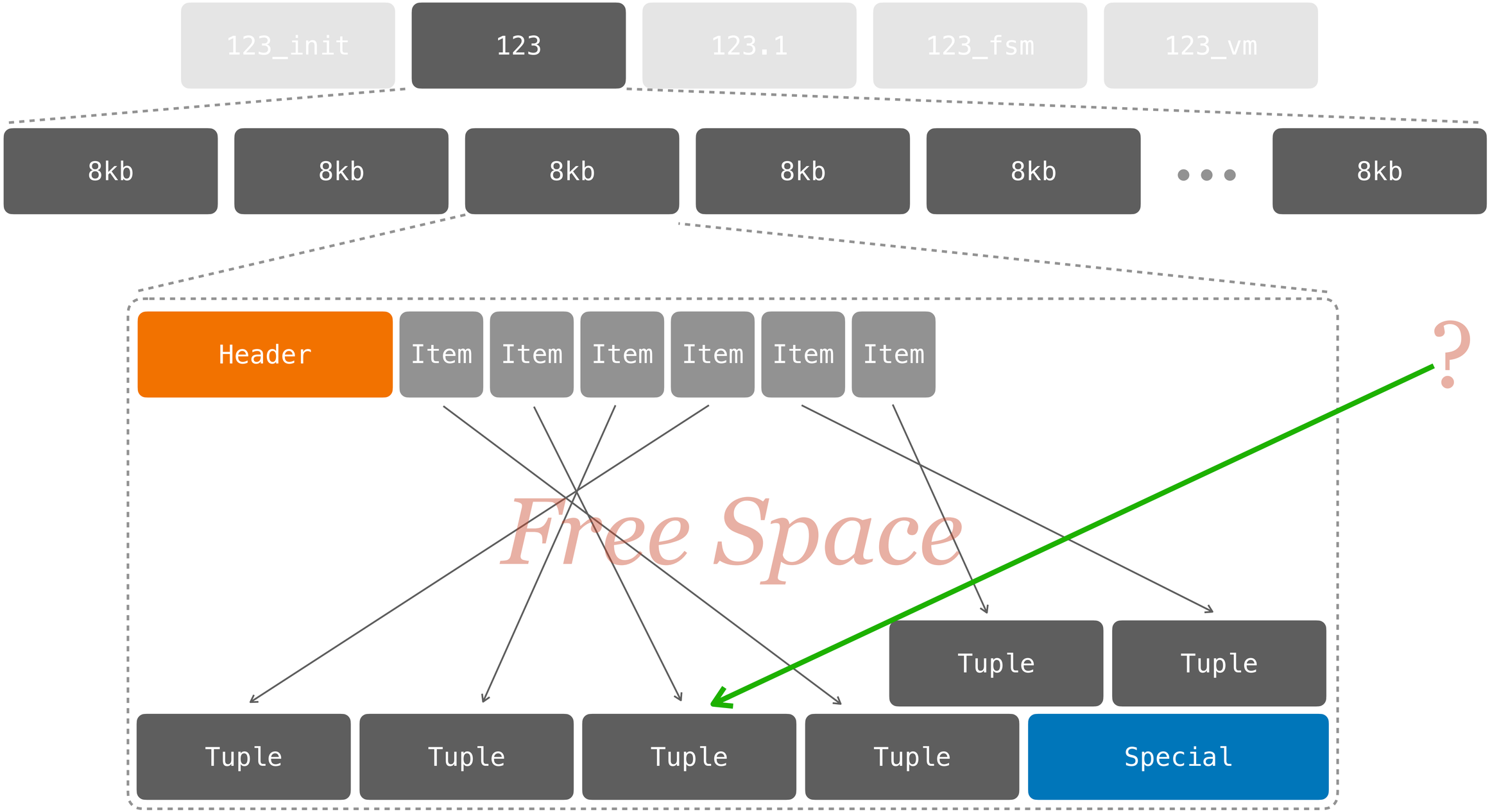
TOAST

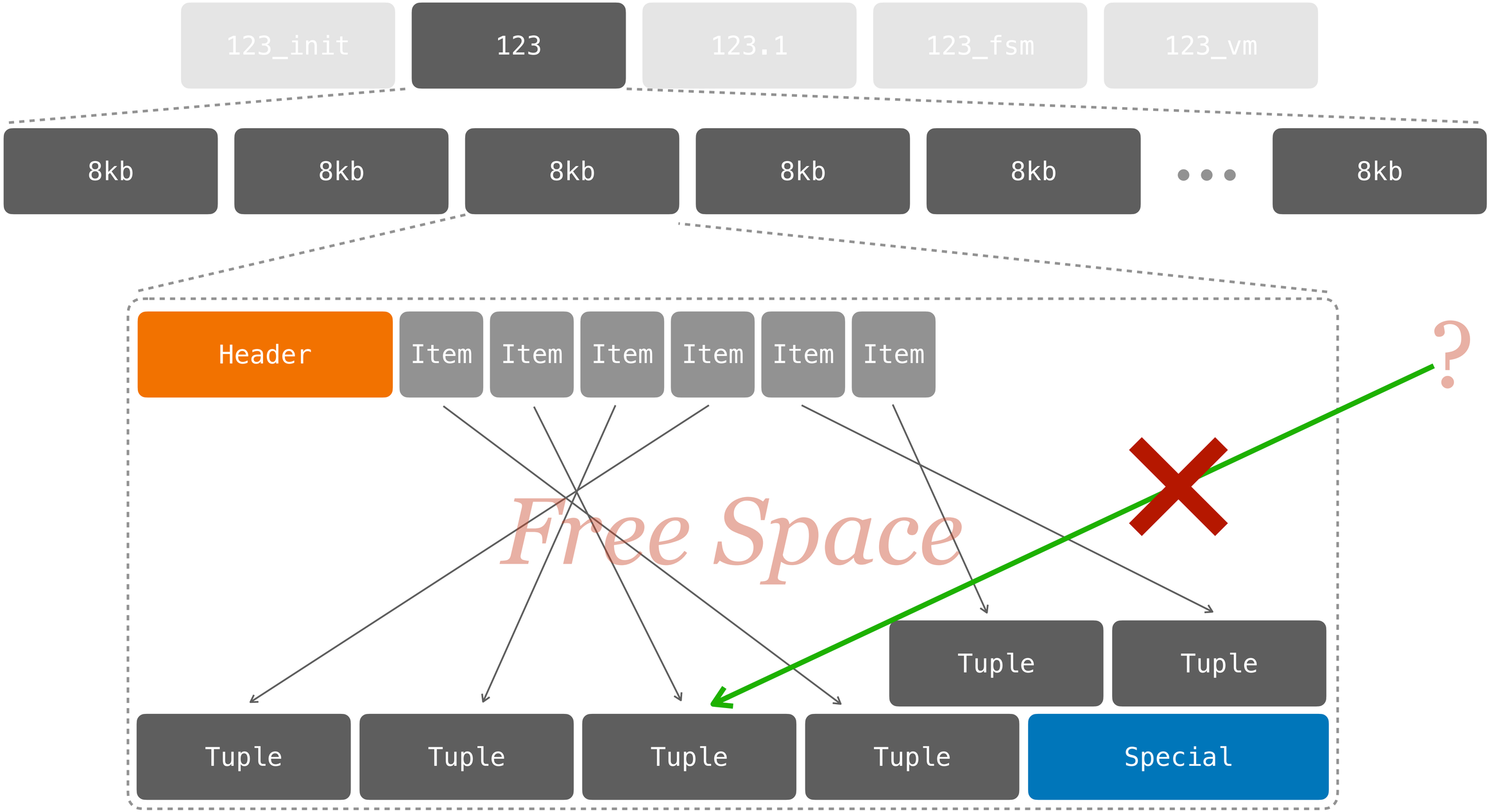
The Oversized Attribute Storage Tech.

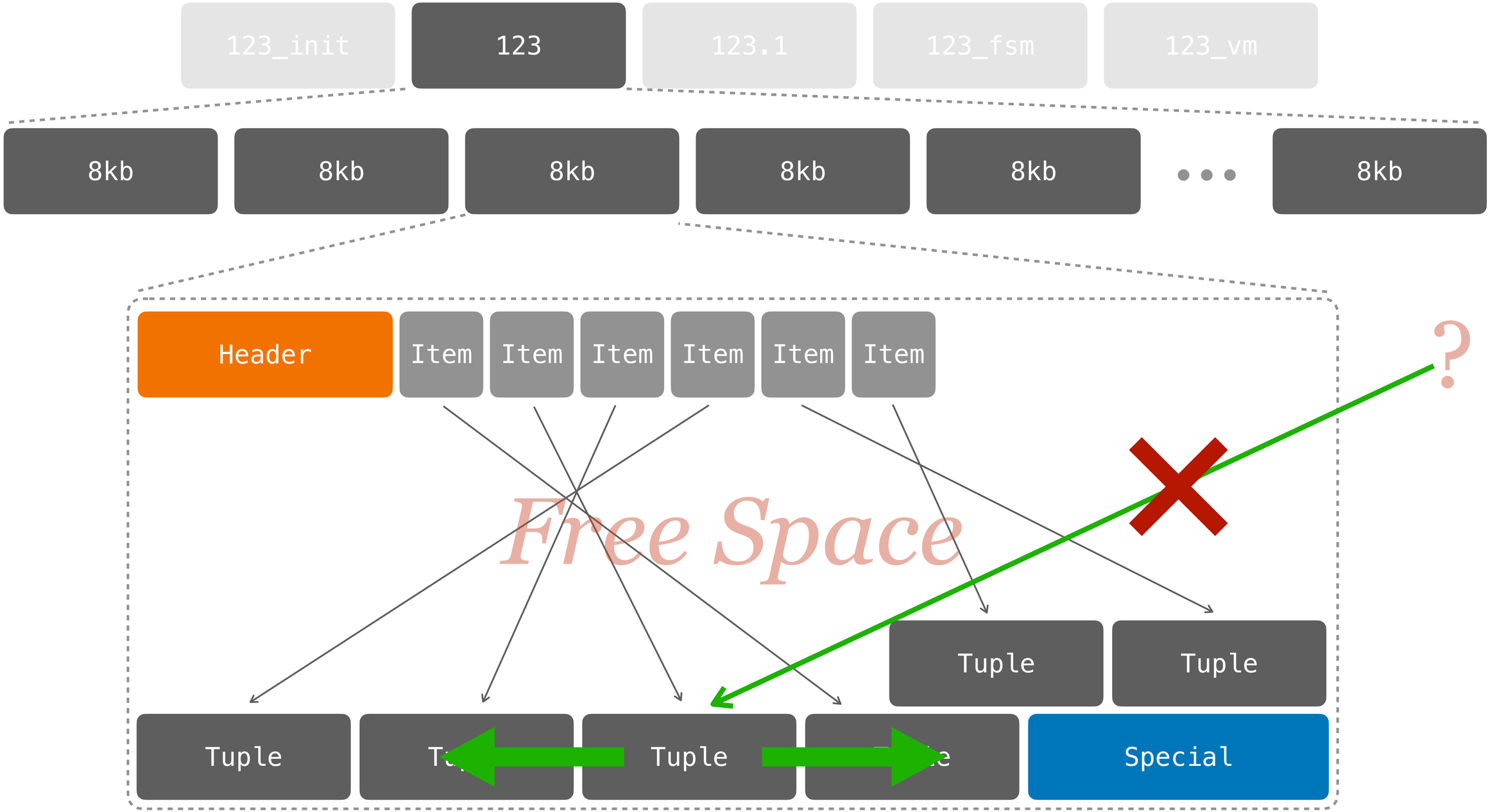
Moving attributes out-of-band

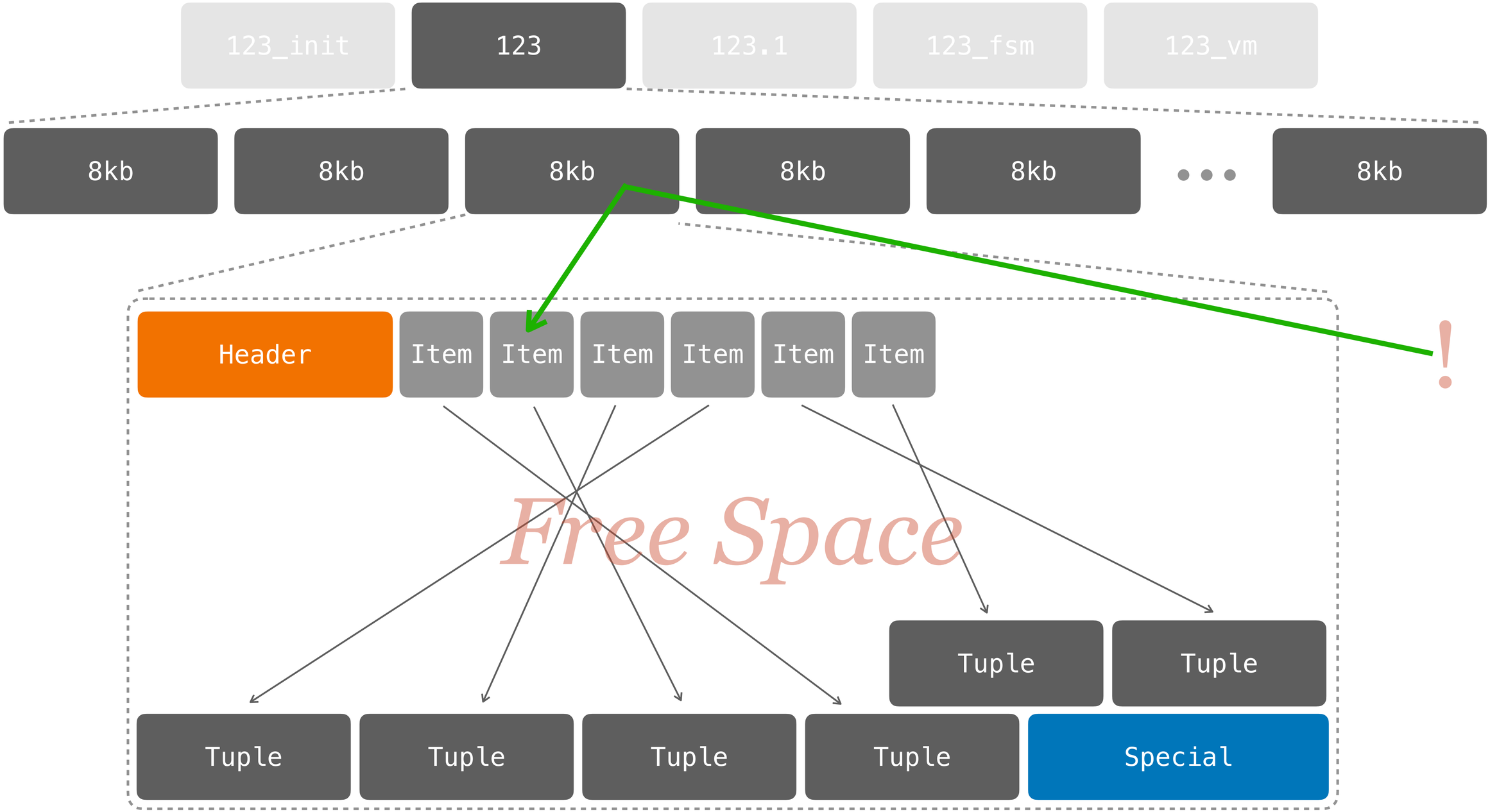
Compression

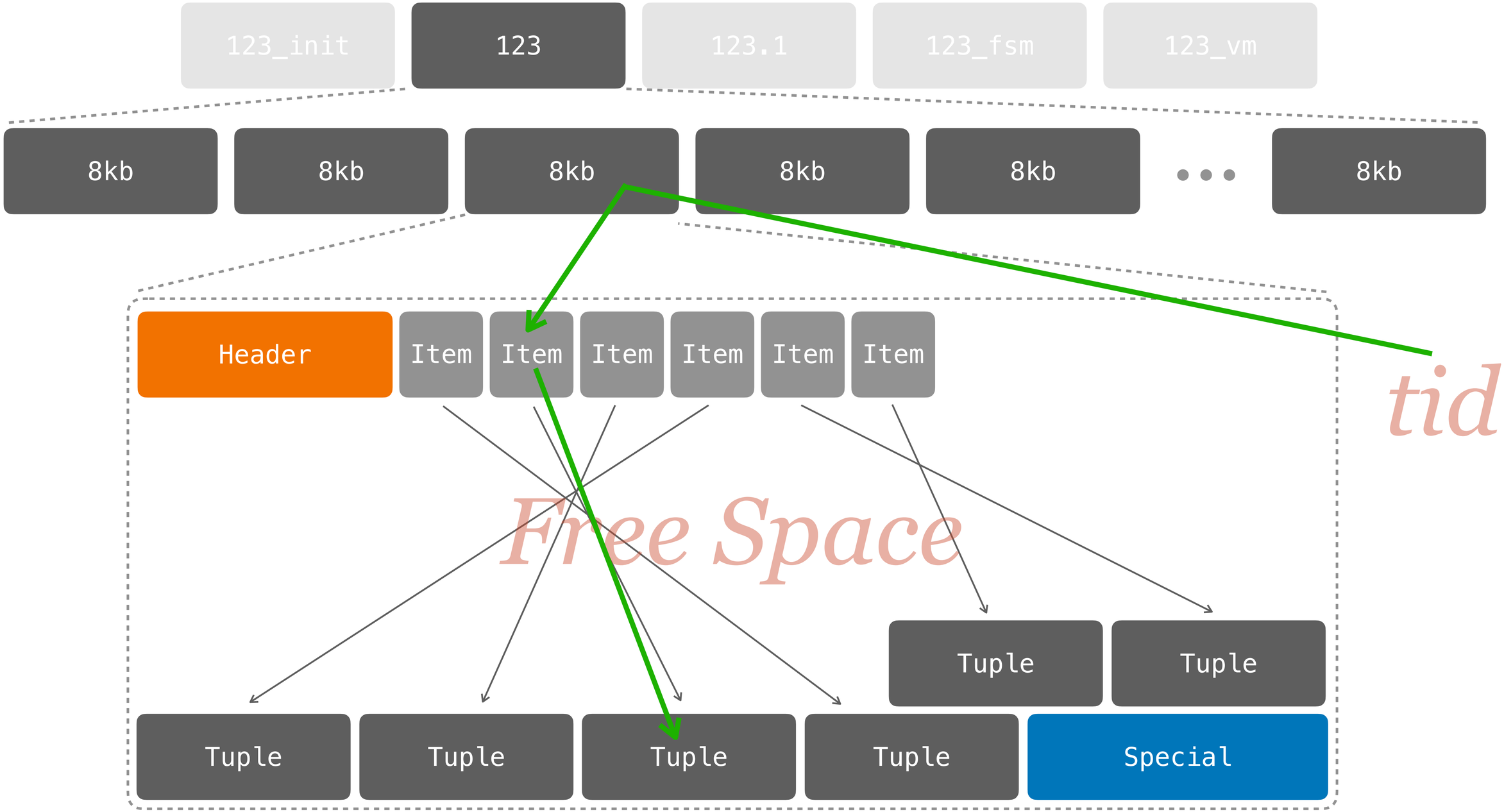
Splits up large values into chunks

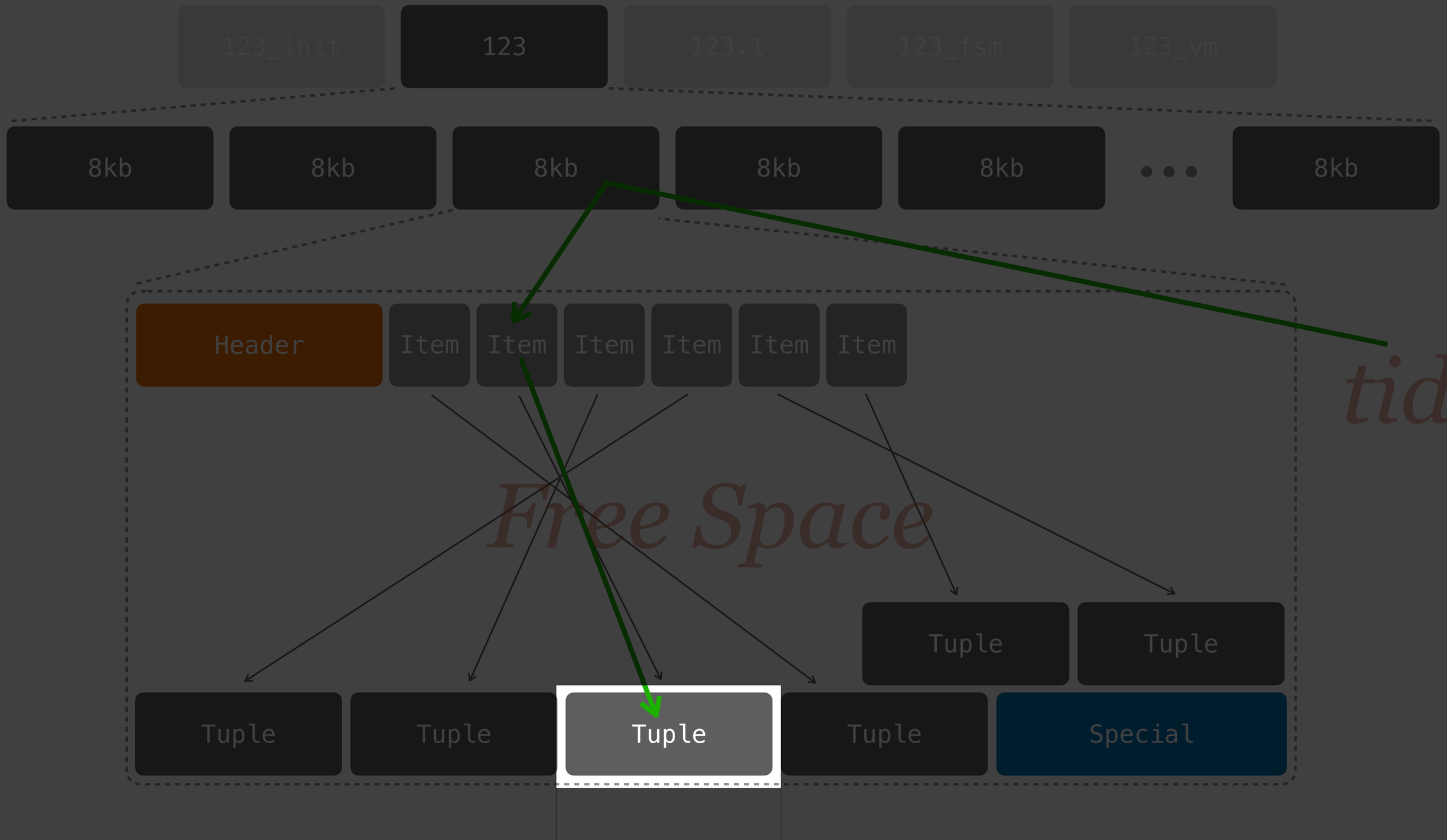

123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb
Header
Item
Item
Item
Item
Item
Item
?
Free Space
Tuple
Tuple
Tuple
Tuple
Tuple
Tuple
Special

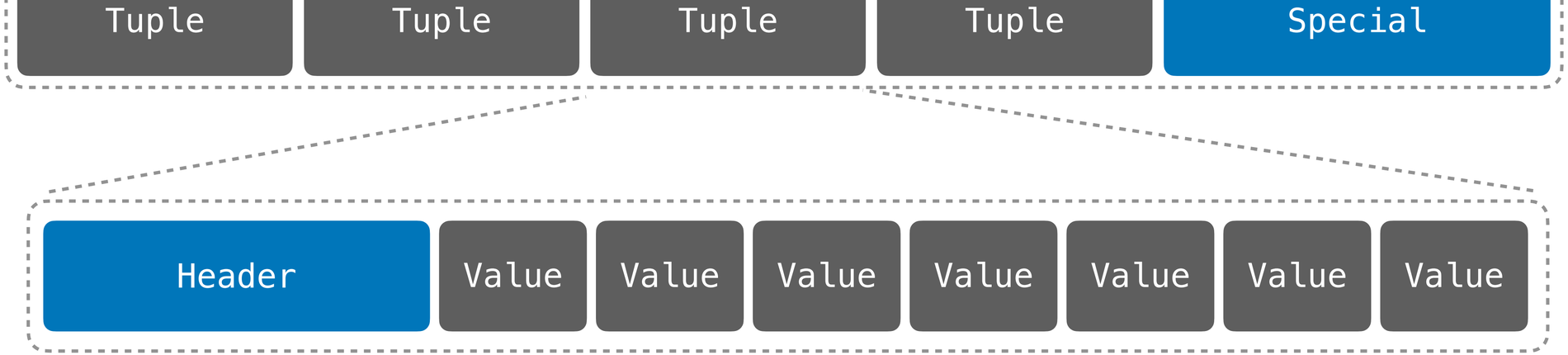

123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb
Header
Item
Item
Item
Item
Item
Item
?
Free Space
Tuple
Tuple
Tuple
Tuple
Tuple
Tuple
Special

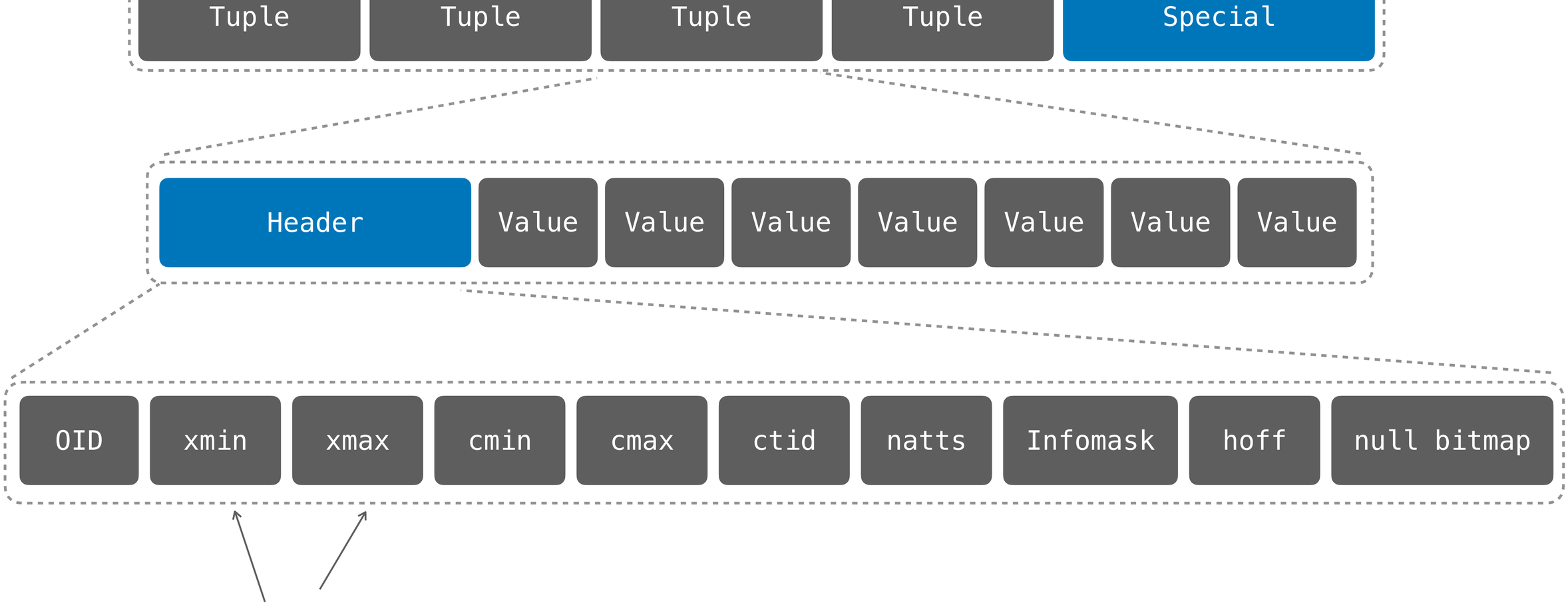

123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb
Header
Item
Item
Item
Item
Item
Item
?
Free Space
Tuple
Tuple
Tuple
Tuple
Special


123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb
Header
Item
Item
Item
Item
Item
Item
!
Free Space
Tuple
Tuple
Tuple
Tuple
Tuple
Tuple
Special

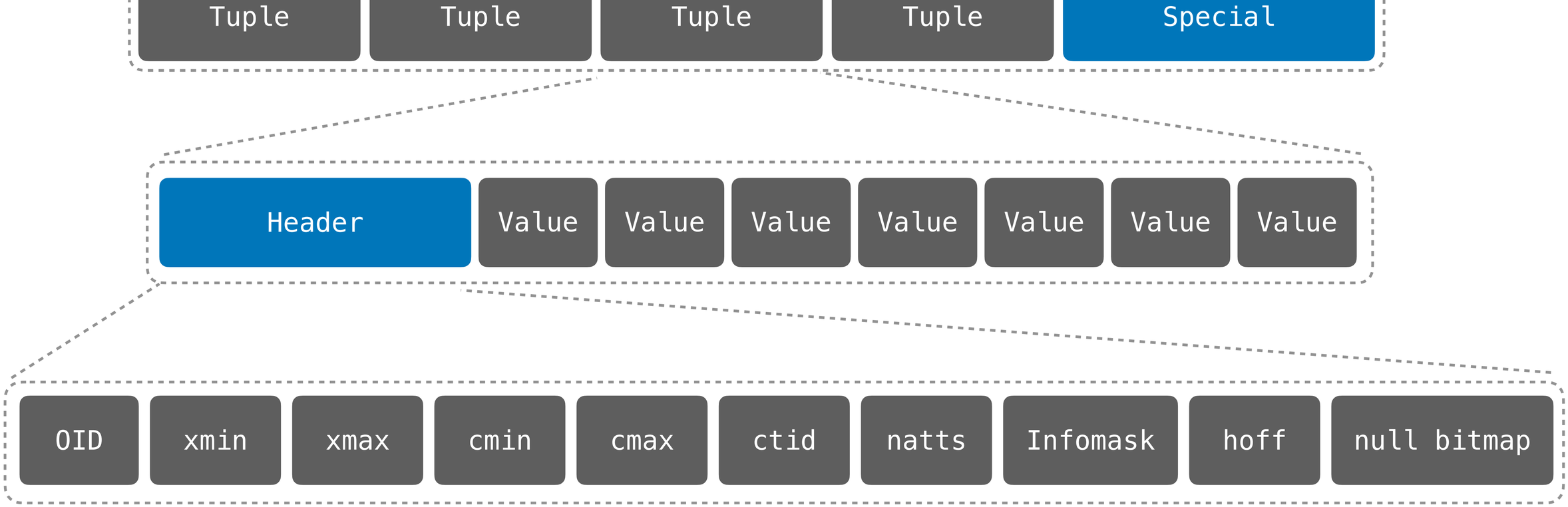

123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb
Header
Item
Item
Item
Item
Item
Item
tid
Free Space
Tuple
Tuple
Tuple
Tuple
Tuple
Tuple
Special


123_init
123
123.1
123_fsm
123_vm
8kb
8kb
8kb
8kb
8kb
8kb
Header
Item
Item
Item
Item
Item
Item
tid
Free Space
Tuple
Tuple
Tuple
Tuple
Tuple
Tuple
Special

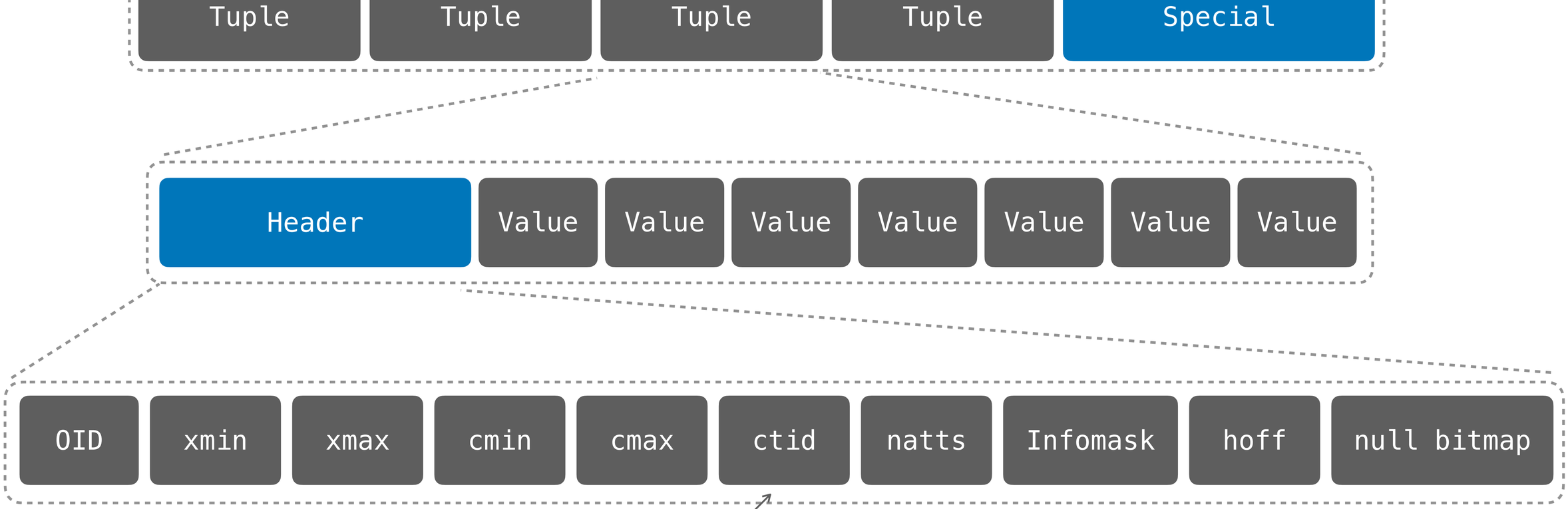

Tuple
Tuple
Tuple
Tuple
Special
Header
Value
Value
Value
Value
Value
Value
Value



Creation and Deletion Transaction IDs



Bitmap representing which columns can contain NULLs



Pointer to the next updated version of the tuple

..next version?

MVCC

Multiversion Concurrency Control

Readers never block writers

Writers never block readers

Updates are Inserts + Deletes

MVCC

Need to reclaim dead rows

Vacuum and autovacuum

Everyones favorite thing to hate

Concurrent with database operations

```
postgres=# create table t (a integer);
CREATE TABLE
postgres=# insert into t values (1);
INSERT 0 1
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |    0 | 1
(1 row)
```

```
postgres=# create table t (a integer);
CREATE TABLE
postgres=# insert into t values (1);
INSERT 0 1
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |    0 | 1
(1 row)
```

```
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |    0 | 1
(1 row)
```

```
postgres=# create table t (a integer);
CREATE TABLE
postgres=# insert into t values (1);
INSERT 0 1
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |    0 | 1
(1 row)
```

```
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |    0 | 1
(1 row)
```

```
postgres=# begin;
BEGIN
postgres=*# update t set a = 2;
UPDATE 1
postgres=*# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  751 |    0 | 2
(1 row)
```

```
postgres=# create table t (a integer);
CREATE TABLE
postgres=# insert into t values (1);
INSERT 0 1
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |    0 | 1
(1 row)
```

```
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |    0 | 1
(1 row)
```

```
postgres=# begin;
BEGIN
postgres=*# update t set a = 2;
UPDATE 1
postgres=*# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  751 |    0 | 2
(1 row)
```

```
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |  751 | 1
(1 row)
```

```
postgres=# create table t (a integer);
CREATE TABLE
postgres=# insert into t values (1);
INSERT 0 1
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |    0 | 1
(1 row)
```

```
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |    0 | 1
(1 row)
```

```
postgres=# begin;
BEGIN
postgres=*# update t set a = 2;
UPDATE 1
postgres=*# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  751 |    0 | 2
(1 row)
```

```
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |  751 | 1
(1 row)
```

```
postgres=*# commit;
COMMIT
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  751 |    0 | 2
(1 row)
```

```
postgres=# create table t (a integer);
CREATE TABLE
postgres=# insert into t values (1);
INSERT 0 1
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |    0 | 1
(1 row)
```

```
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |    0 | 1
(1 row)
```

```
postgres=# begin;
BEGIN
postgres=*# update t set a = 2;
UPDATE 1
postgres=*# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  751 |    0 | 2
(1 row)
```

```
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  750 |  751 | 1
(1 row)
```

```
postgres=*# commit;
COMMIT
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  751 |    0 | 2
(1 row)
```

```
postgres=# select xmin,xmax,* from t;
 xmin | xmax | a
------+------+---
  751 |    0 | 2
(1 row)
```

Transaction ID Wraparound

Transaction IDs are 32-bit

4 billion transactions can whoosh past quickly

Wraparound would cause time-travel

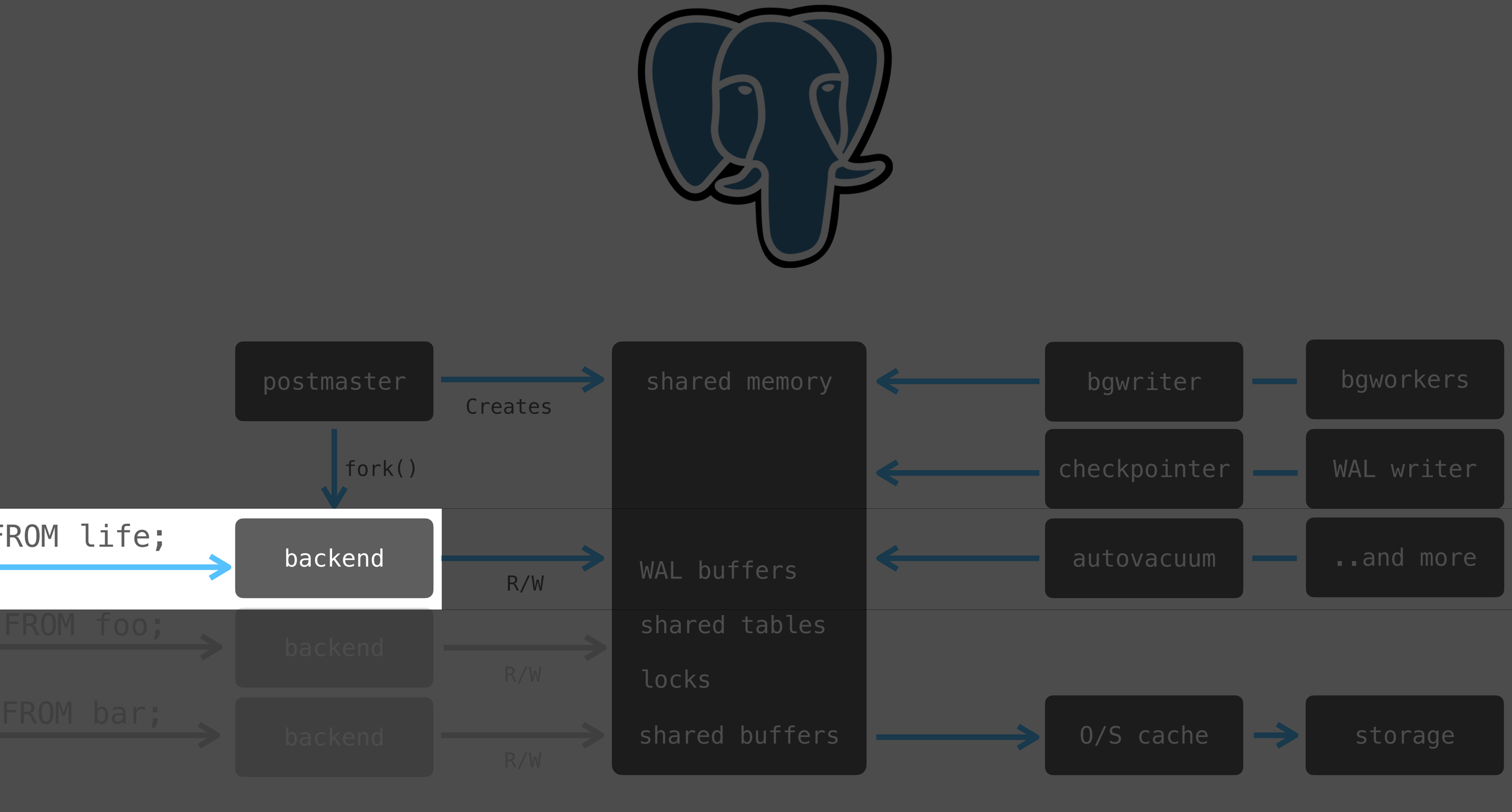
No shortage of horror stories..

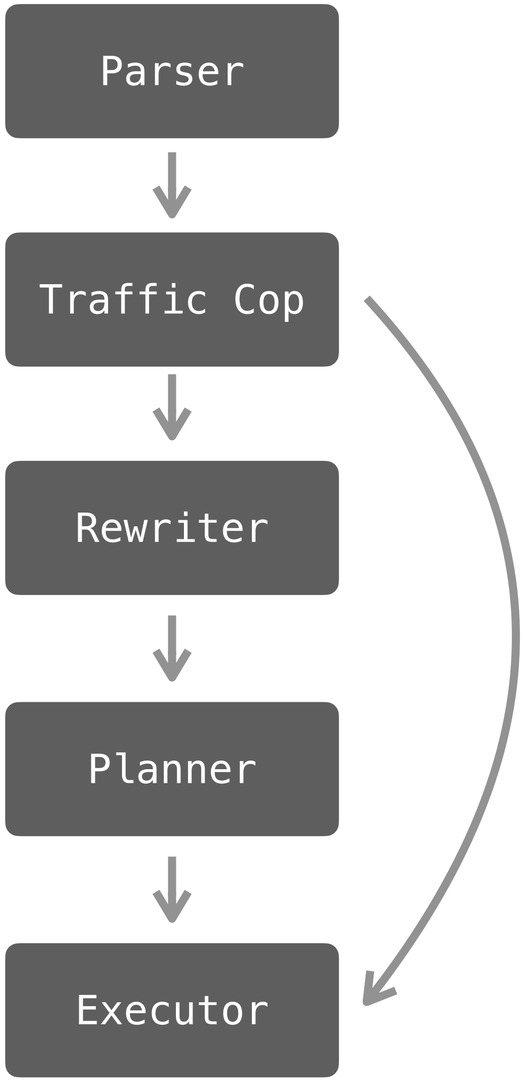
Freezing

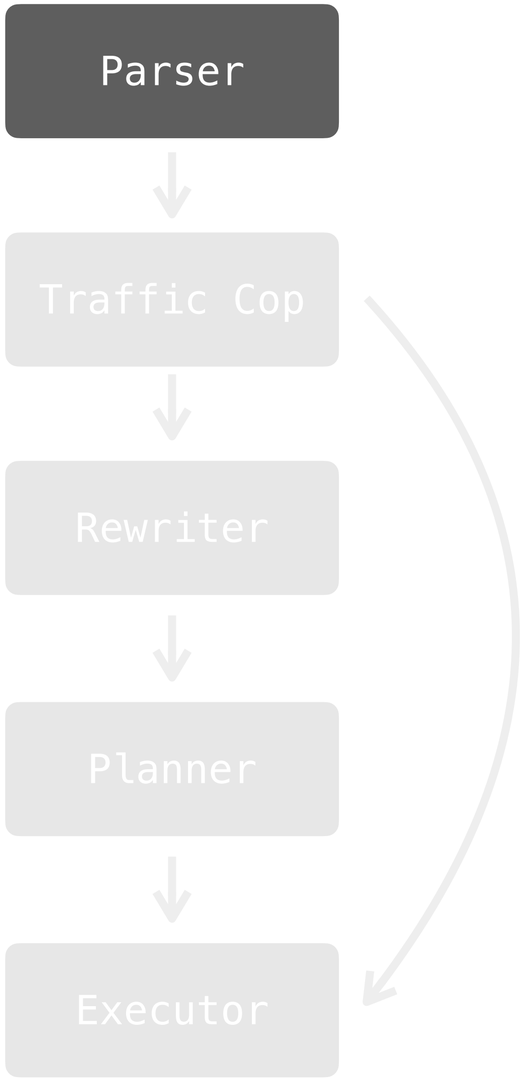
Tuples visible to all don't need xmin

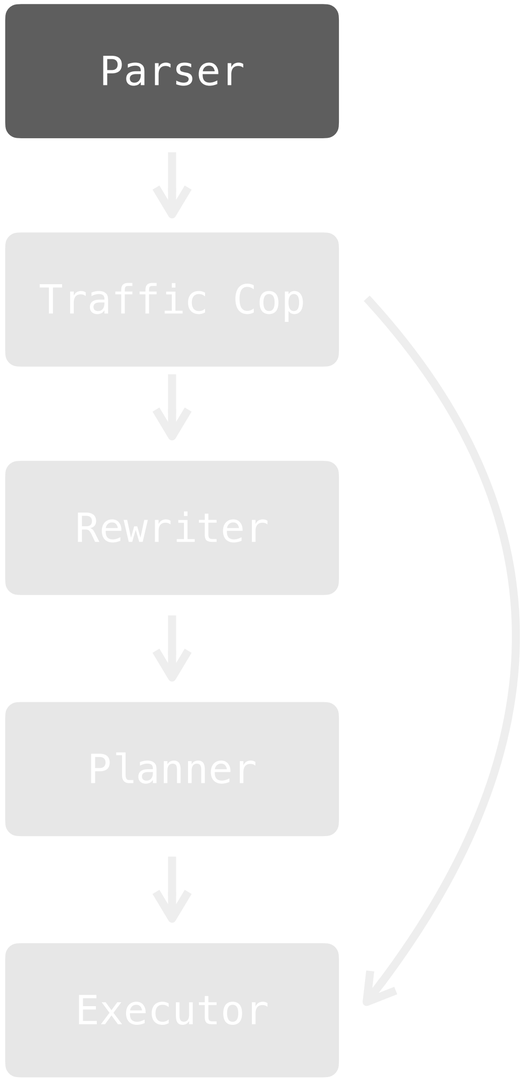
VACUUM freezes such tuples

xmax in tuples not visible to anyone can be reused


postmaster
Creates
shared memory
bgwriter
bgworkers
fork()
checkpointer
WAL writer
FROM life;
backend
R/W
WAL buffers
autovacuum
..and more
FROM foo;
backend
R/W
shared tables
locks
FROM bar;
backend
R/W
shared buffers
O/S cache
storage

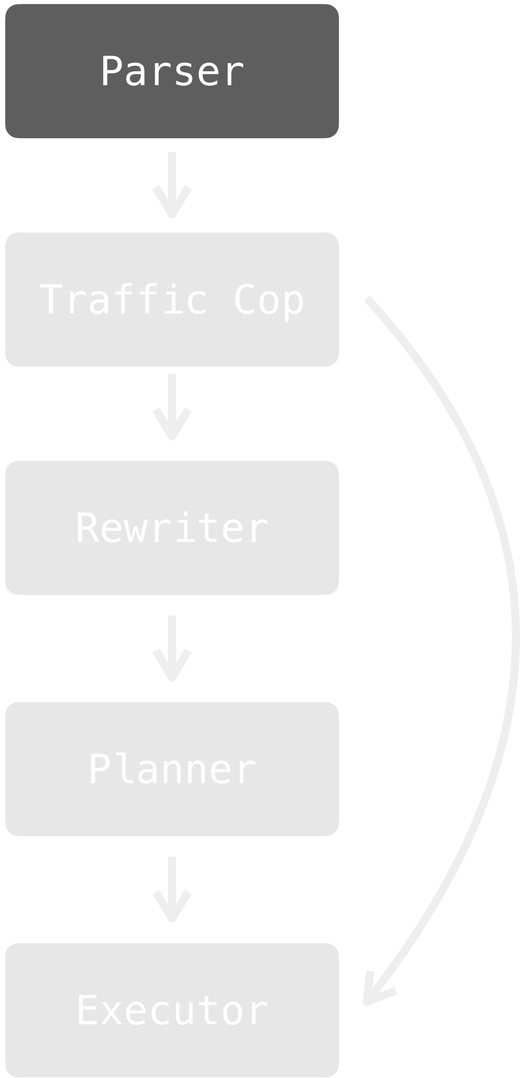

Parser
Traffic Cop
Rewriter
Planner
Executor


Parser
Traffic Cop
Rewriter
Planner
Executor




```
postgres=# set debug_print_parse to on;
SET
```

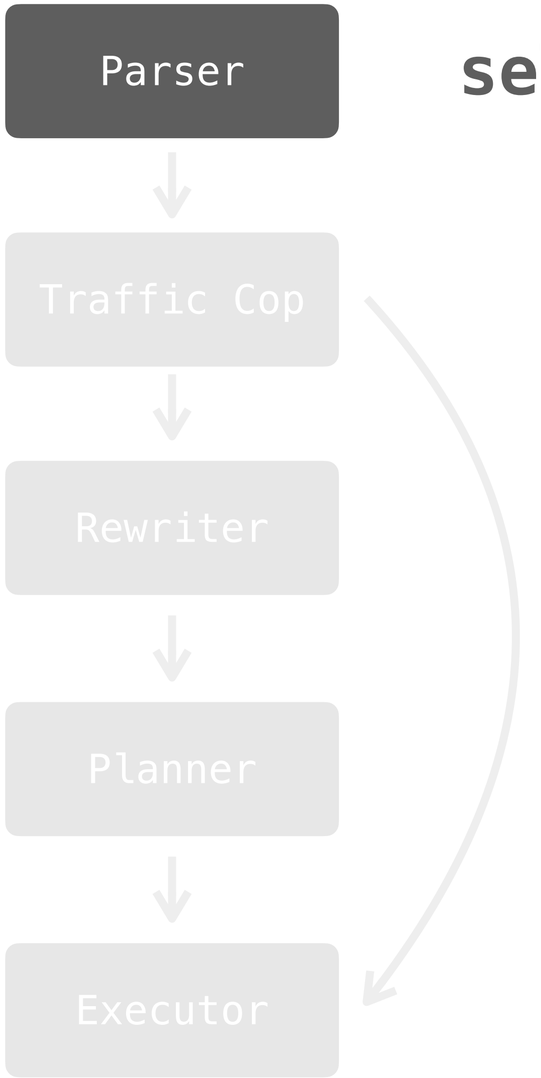
Prints parsetree into log




```
postgres=# set debug_print_parse to on;
SET
```

Prints parsetree into log

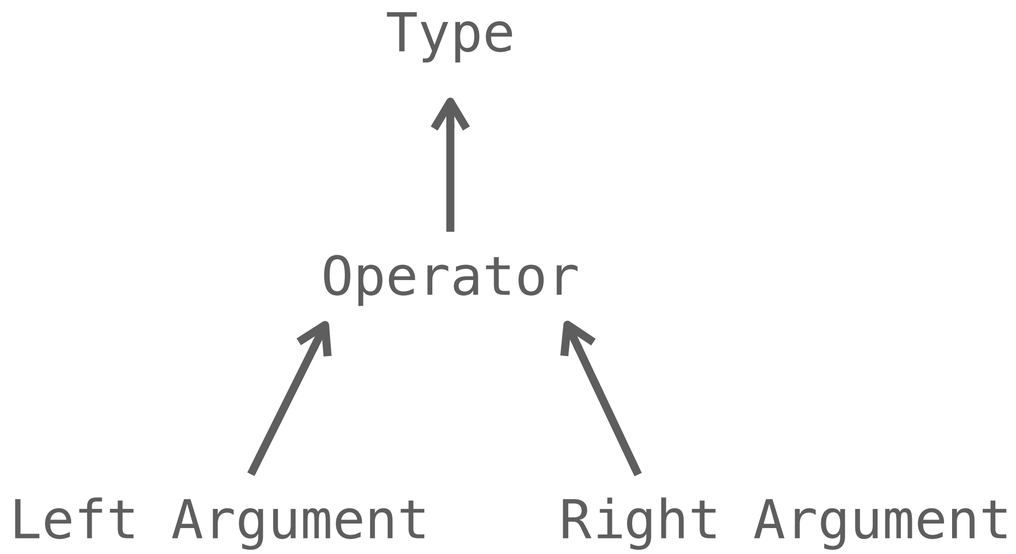
GUC

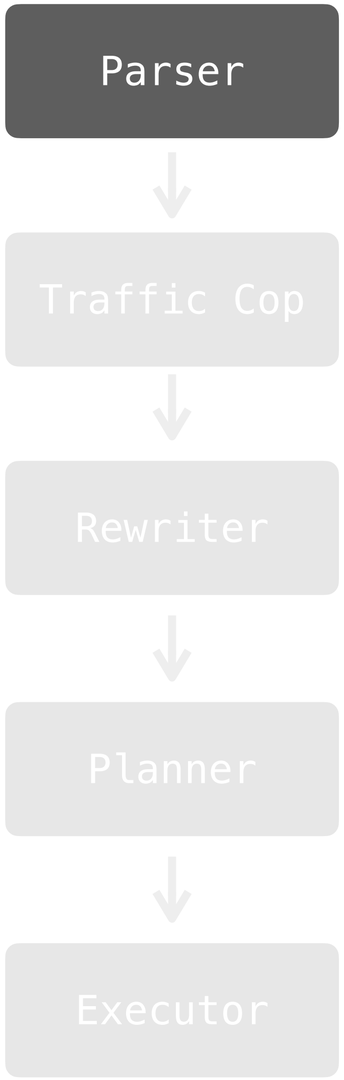



```
select t.a,tt.b from t join tt on (t.a = tt.b);
```


```
select t.a,tt.b from t join tt on (t.a = tt.b);
```

Join Tree Qualification

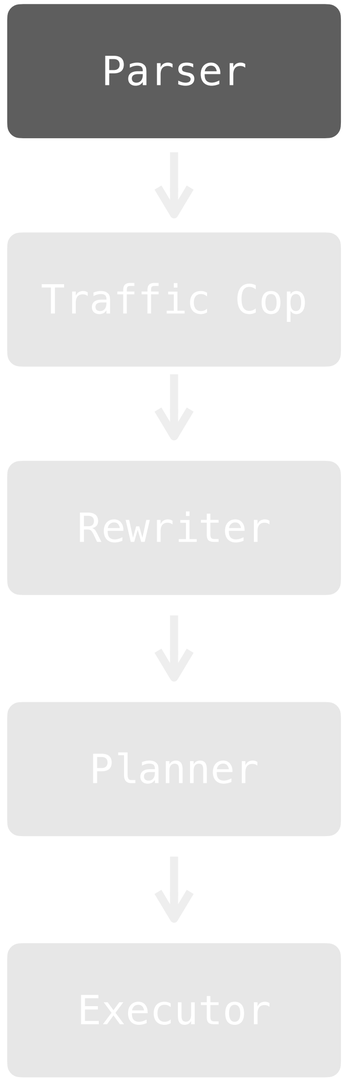





```
select t.a,tt.b from t join tt on (t.a = tt.b);
```

Qualification Operator

```
{OPEXPR
:opno 96
:opfuncid 65
:opresulttype 16
```




```
select t.a,tt.b from t join tt on (t.a = tt.b);
```

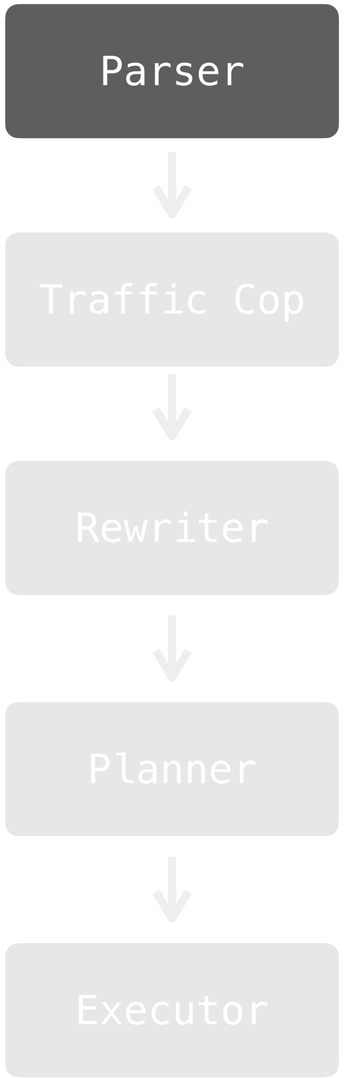
Qualification Operator

```
{OPEXPR
:opno 96
:opfuncid 65
:opresulttype 16
```

```
postgres=# select oprname from pg_operator where oid = 96;
 oprname
---------
 =
(1 row)
```

```
postgres=# select prosrc from pg_proc where oid = 65;
 prosrc
--------
 int4eq
(1 row)
```

```
postgres=# select typname from pg_type where oid = 16;
 typname
---------
 bool
(1 row)
```




```
select t.a,tt.b from t join tt on (t.a = tt.b);
```

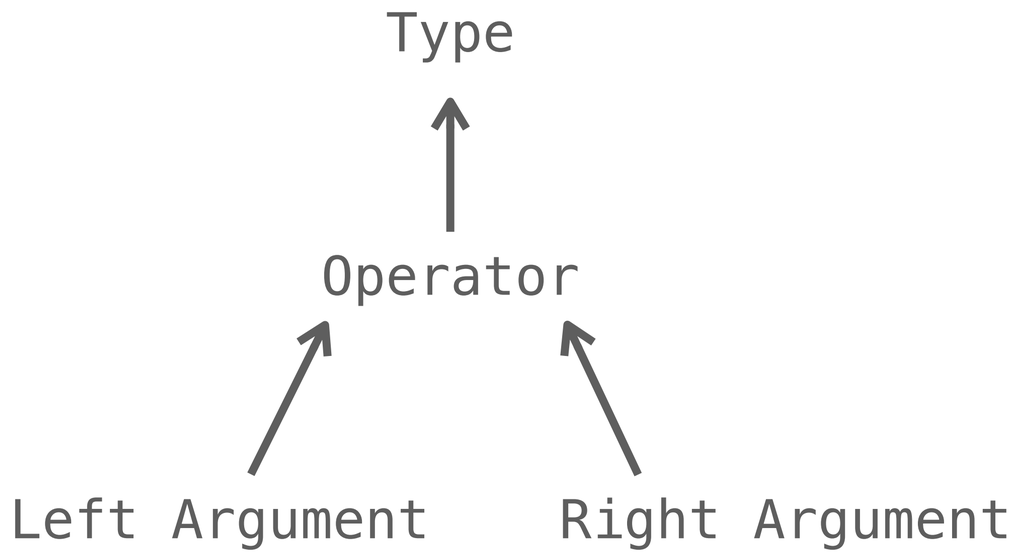
Qualification Arguments

```
:args (
   {VAR
   :varno 1
   :varattno 1
   :vartype 23
   :location 35
   }
   {VAR
   :varno 2
   :varattno 1
   :vartype 23
   :location 41
   }
)
```

```
postgres=# select typname from pg_type where oid = 23;
 typname
---------
 int4
(1 row)
```

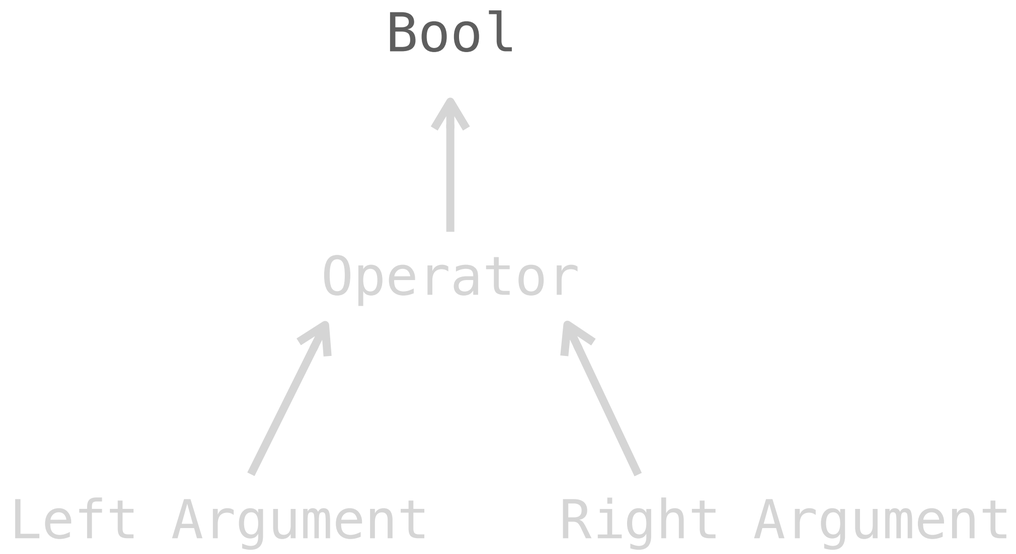

```
select t.a,tt.b from t join tt on (t.a = tt.b);
```

Join Tree Qualification



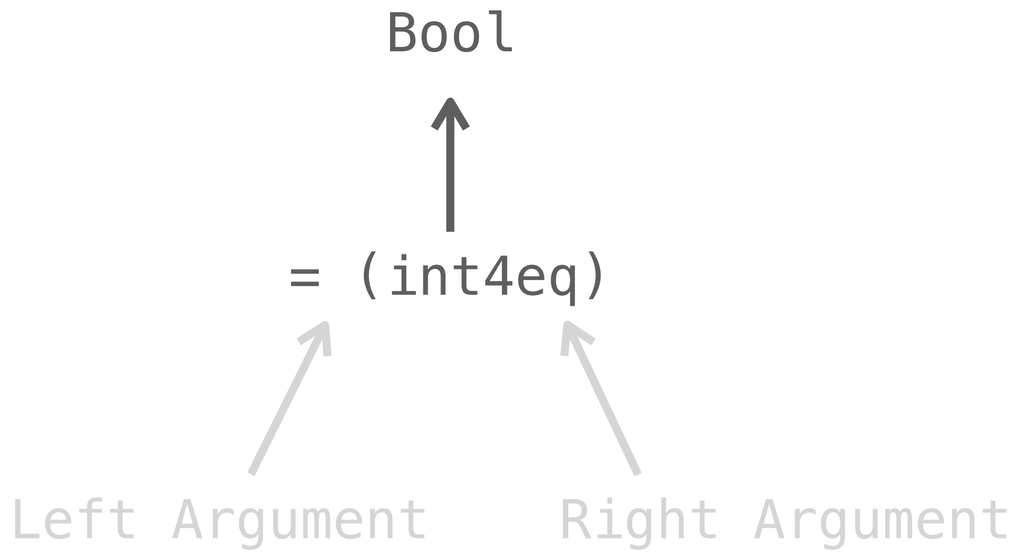

```
select t.a,tt.b from t join tt on (t.a = tt.b);
```

Join Tree Qualification



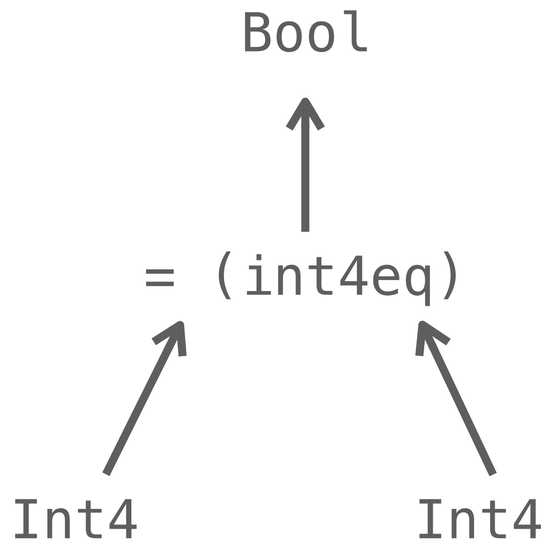

```
select t.a,tt.b from t join tt on (t.a = tt.b);
```

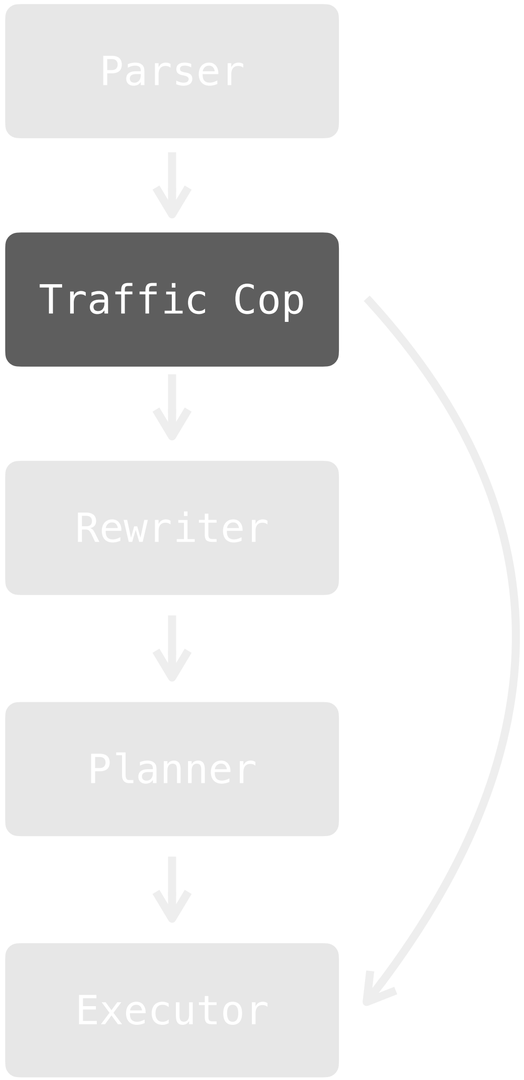
Join Tree Qualification




```
select t.a,tt.b from t join tt on (t.a = tt.b);
```

Join Tree Qualification





Traffic Cop

Handles Utility Queries

`CREATE, ALTER ..`

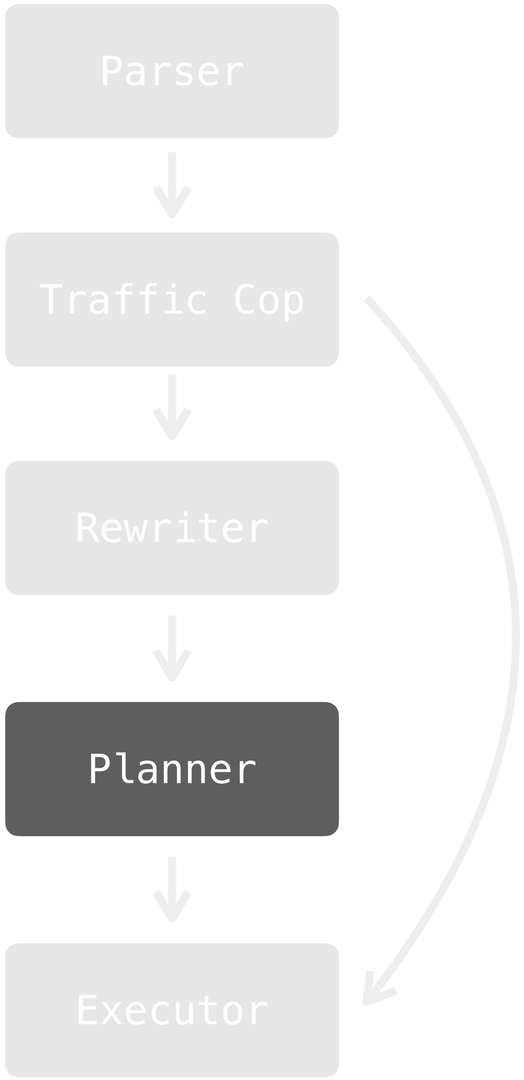
Not planned

Rewriter

Substitutes views with sub-queries

Expands rules

Transformations



Planner

Cost-estimate based

Generates a tree of plan nodes

Join order selection

Genetic Algorithm

Planner

Parser

Traffic Cop

iter

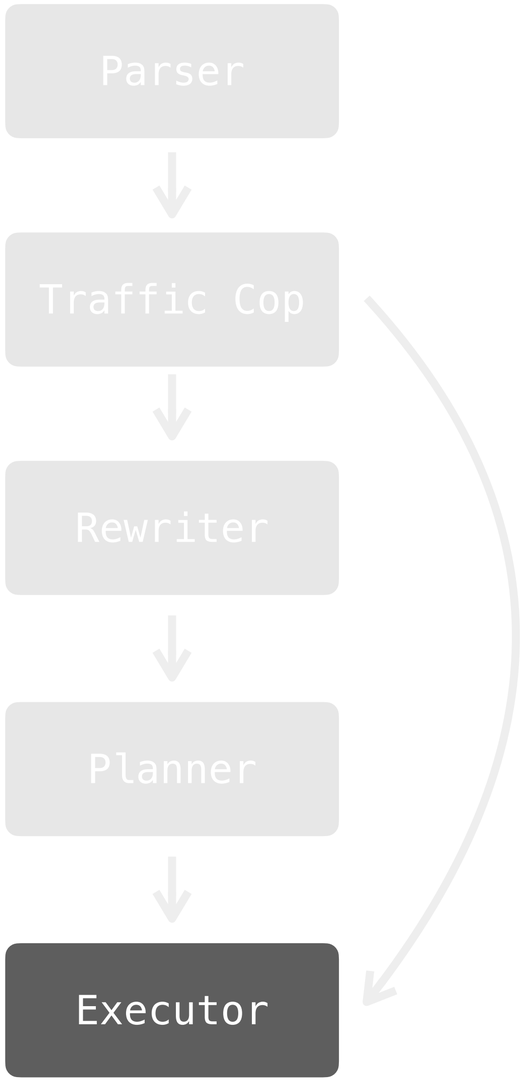
ner

Cost-estimate based

rates a

hoices

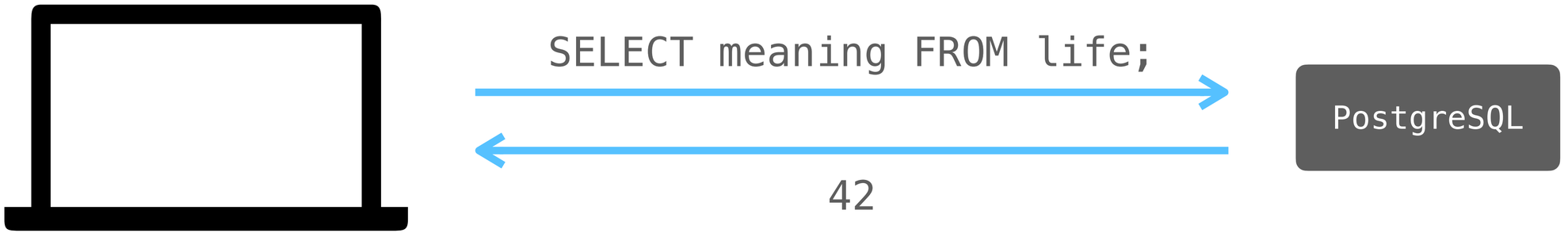
Algorith

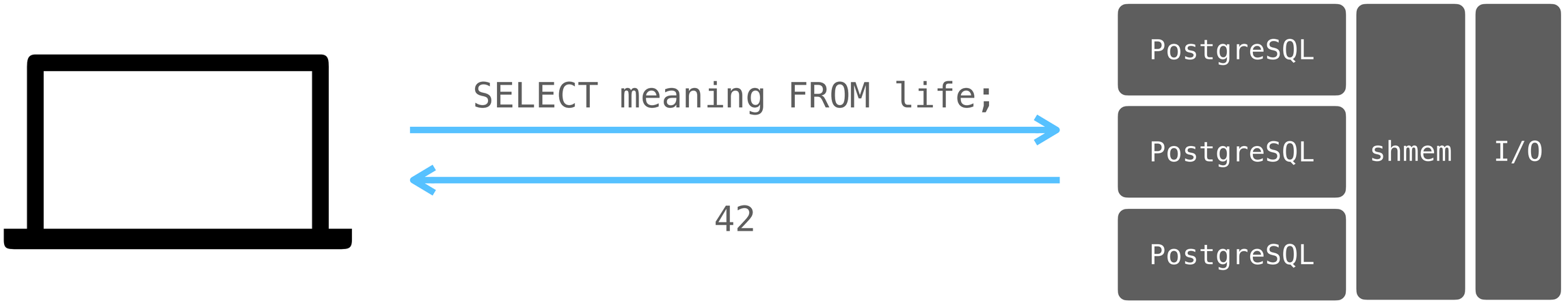


Executor

Each node produce output to next node

Flattens tree into expression steps


SELECT meaning FROM life;
PostgreSQL
42


SELECT meaning FROM life;
42
PostgreSQL
PostgreSQL
PostgreSQL
shmem
I/O

TL;DR

Postgres is complicated

Microsoft is a great place to learn it

Future sessions will dive deeper into different subsystems

Microsoft

Thank you!

dgustafsson@microsoft.com

Or find me on Teams