credativ

Beyond work_mem Myths

A Source Code Guided Tour Through PostgreSQL Memory Usage (version 2)

Josef Machytka <josef.machytka@credativ.de>

2026-04-22 German PostgreSQL Conference

credativ GmbH

- Founded 1999 in Jülich, Germany
- Close ties to Open-Source Community
- More than 40 Open-Source experts
- Consulting, development, training, support (3rd-level / 24x7)
- Open-Source infrastructure with Linux, Kubernetes, Proxmox
- Open-Source databases with PostgreSQL
- DevSecOps with Ansible, Puppet, Terraform and others
- Since 2025 independent owner-managed company again

credativ.de

Josef Machytka

credativ

- Professional Service Consultant - PostgreSQL specialist at credativ GmbH
- 33+ years of experience with different databases
- PostgreSQL (13y), BigQuery (7y), Oracle (15y), MySQL (12y), Elasticsearch (5y), MS SQL (5y)
- 10+ years of experience with Data Ingestion pipelines, Data Analysis, Data Lake and Data Warehouse
- 3+ years of practical experience with different LLMs / AI / ML including architecture and principles
- From Czechia, living now 12 years in Berlin

- LinkedIn: linkedin.com/in/josef-machytka
- Medium: medium.com/@josef.machytka
- YouTube: youtube.com/@JosefMachytka

- GitHub: github.com/josmac69/conferences_slides
- ResearchGate: researchgate.net/profile/Josef-Machytka

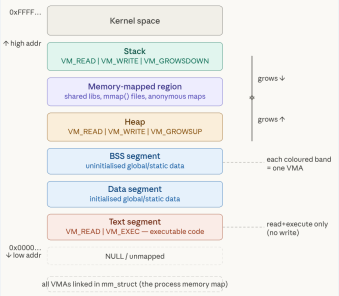
All My Slides:

Recoded talks:

Process Address Space in Linux

- Memory for a process is allocated as a massive Virtual Address Space
- Linux kernel divides this address space into VMAs
- A Virtual Memory Area (VMA) is a continuous range of virtual addresses
- The idea is to have ranges with identical attributes

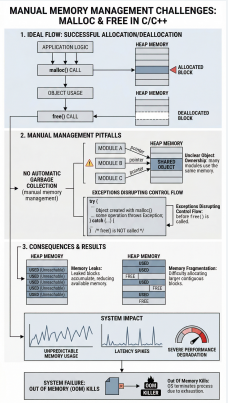
- Each VMA has specific properties and permissions
  - **Text Segment** - compiled code of the program (R-X)
  - **Data Segment** - initialized global & static vars (R-W)
  - **BSS Segment** - uninitialized global & static vars (zeroed at startup) (R-W) historical name "Block Started by Symbol"
  - **Heap** - dynamic memory allocated during runtime (calls malloc()) (R-W)
  - **Memory Mapping** - shared libraries or files mapped into memory (R-W-X)
  - **Stack** - local vars, func params, return addresses - grows downwards



Images and diagrams created by Gemini 3.1 Pro

Memory Management in C / C++ Can Be Tricky

- Dynamic allocation / deallocation using malloc / free
- No Automatic Garbage Collection = manual memory management
- Unclear Object Ownership - many modules use the same memory
- Exceptions Disrupting Control Flow - before delete is called
- Results in memory leaks, big memory fragmentation
- Severe Performance Degradation, Out Of Memory Kills
- Unpredictable Memory Usage, Latency spikes

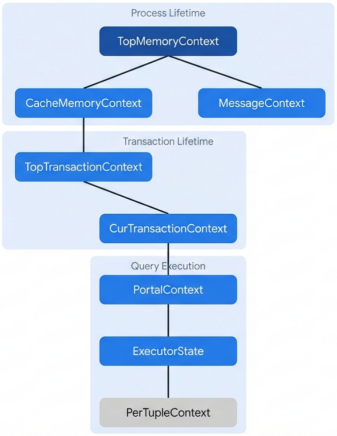


PostgreSQL Memory Management

credativ

- Uses arena-based - context memory management
- Each context is associated with specific operation
- Context is deleted after operation completes
- Developer does not need to track specific objects

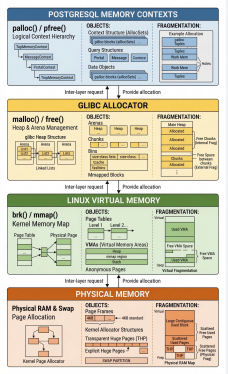
- Memory Contexts are hierarchical
  - **TopMemoryContext** - created on start of backend process
  - **CacheMemoryContext** - metadata of relations, catalog, query plans
  - **MessageContext** - raw command string, planning & parsing data
  - **TopTransactionContext** - lock states, transaction handlers
  - **PortalContext** - current active statement or cursor
  - **ExecutorContext** - memory for joins, sorts, hashes, partial results
  - **PerTupleContext** - recycled after tuple is processed to avoid bloats



Memory Context vs Physical Memory

- PostgreSQL Memory Contexts are Virtual Memory Areas
- Visible only inside PostgreSQL through system objects
- View `pg_backend_memory_contexts` for current session
- Func `pg_log_backend_memory_contexts` for other sessions

- On Linux - physical memory visible in `/proc/[PID]/smaps`
  - **Linux libraries** - mapped to specific files
  - **Shared memory Areas** - mapped to /dev/shm or /dev/zero
  - **anonymous memory** - for sorting, hashing, etc. (BSS segment)
  - **heap** - parsing, planning, small data retrieval (Heap segment)



How Much Memory Does PostgreSQL Use?

credativ

- PC 32GB memory, PostgreSQL 18, shared_buffers=8GB, effective_cache_size=24GB, work_mem=64MB
- Multiple sessions with/without parallelism
- What do the ”top” and ”free” commands show?
- Panic - machine must be dying - memory is clearly totally used - right?

```
## top command
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
190747 postgres  20   0 8701868   8.1g   8.1g S   0.0  26.2   0:48.69 postgres: postgres postgres 172.18.0.1(40278) idle
206119 postgres  20   0 8702808   4.7g   4.7g S   0.0  15.2   0:21.81 postgres: postgres postgres 172.18.0.1(44010) idle
219065 postgres  20   0 8802384   8.2g   8.1g S   0.0  26.6   2:56.64 postgres: postgres postgres 172.18.0.1(52912) idle
228832 postgres  20   0 8709912   3.4g   3.4g S   0.0  11.1   0:11.10 postgres: postgres postgres 172.18.0.1(60090) idle
230390 postgres  20   0 8701340   8.1g   8.1g S   0.0  26.3   0:51.89 postgres: postgres postgres 172.18.0.1(44802) idle

## free command
               total        used        free      shared  buff/cache   available
Mem:            31Gi        15Gi       698Mi       9.3Gi        24Gi        15Gi
Swap:           31Gi       4.5Gi        26Gi
Comm:           46Gi        67Gi       -21Gi
```

System Sources for Memory Usage

- We must use proc filesystem objects - /proc/meminfo, /proc/[PID]/status, /proc/[PID]/smaps
- For our tests smaps fits best, contains multiple entries for each memory region
- Header contains memory region start and end address, permissions, path to the file
- Some memory regions do not contain any path

```
557b3de3c000-557b3df0e000 r--p 00000000 fd:02 14426095                   /usr/lib/postgresql/18/bin/postgres
Size:                840 kB
KernelPageSize:        4 kB
MMUPageSize:           4 kB
Rss:                 544 kB
Pss:                 272 kB
Pss_Dirty:             0 kB
Shared_Clean:        544 kB
Shared_Dirty:          0 kB
Private_Clean:         0 kB
Private_Dirty:         0 kB
Referenced:          544 kB
Anonymous:             0 kB
LazyFree:              0 kB
AnonHugePages:         0 kB
ShmemPmdMapped:        0 kB
FilePmdMapped:         0 kB
Shared_Hugetlb:        0 kB
Private_Hugetlb:       0 kB
Swap:                  0 kB
SwapPss:               0 kB
Locked:                0 kB
```

Reading smaps File Content

- Command pmap can show smaps file content in table form

```
josef@josefx-lin-0 ~ $ sudo pmap -X -p 23319
23319:   postgres: postgres postgres [local] idle
         Address Perm   Offset Device  Inode  Size  Rss  Pss Pss_Dirty Referenced Anonymous KSM LazyFree ShmemPmdMapped FilePmdMapped Shared_Hugetlb Private_Hugetlb Swap SwapPss Locked THPeligible ProtectionKey Mapping
560bbad16000 r--p 00000000 00:44 16258918    876    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/postgresql/17/bin/postgres
560bbae67000 r-xp 000db000 00:44 16258918   5712 3012  891         0       3012         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/postgresql/17/bin/postgres
560bbf23fb00 r--p 006f1000 00:44 16258918   2924 1016  197         0       1016         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/postgresql/17/bin/postgres
560bbf6d6000 r--p 00949000 00:44 16258918    144  144   20        20         36       144   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/postgresql/17/bin/postgres
560bbf6fa000 rw-p 0096d000 00:44 16258918    100  100   59        57        100        88   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/postgresql/17/bin/postgres
560bbf713000 rw-p 00000000 00:00 0            220   60   60        60         60        60   0        0               0             0             0               0    0       0      0           0             0
560bca330000 rw-p 00000000 00:00 0            776  616  263       263        616         0   0        0               0             0             0               0    0       0      0           0             0 [heap]
560bca3f2000 rw-p 00000000 00:00 0            568  408  408       408        408         0   0        0               0             0             0               0    0       0      0           0             0 [heap]
7fe7e4311000 rw-p 00000000 00:00 0            260  120  120       120        120       120   0        0               0             0             0               0    0       0      0           0             0
7fe7e4352000 rw-p 00000000 00:5f              12 1024   88        80         88         0   0        0               0             0             0               0    0       0      0           0             0 /dev/shm/PostgreSQL.3074188214
7fe7e4452000 rw-p 00000000 00:00 0            132  128  128       128        128       128   0        0               0             0             0               0    0       0      0           0             0
7fe7e4473000 rw-s 00000000 00:5f              11   28    4         1          1         4   0        0               0             0             0               0    0       0      0           0             0 /dev/shm/PostgreSQL.2588904430
7fe7e447a000 rw-s 00000000 00:01     2448 146344 3084 1467      1467       3084         0   0        0               0             0             0               0    0       0      0           0             0 /dev/zero (deleted)
7fe7ed364000 r--p 00000000 00:44 16257547   2980  124   36         0        124         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/locale/locale-archive
7fe7ed64d000 ---p 00000000 00:00 0             36   36   11        11          8        36   0        0               0             0             0               0    0       0      0           0             0
7fe7ed656000 r--p 00000000 00:44 16253850       8    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libffi.so.8.1.2
7fe7ed658000 r-xp 00002000 00:44 16253850      24    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libffi.so.8.1.2
7fe7ed65e000 r--p 00008000 00:44 16253850       8    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libffi.so.8.1.2
7fe7ed660000 r--p 00009000 00:44 16253850       4    4    0         0          4         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libffi.so.8.1.2
7fe7ed661000 rw-p 0000a000 00:44 16253850       4    4    0         0          4         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libffi.so.8.1.2
7fe7ed662000 r--p 00000000 00:44 16253859      20    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libgpg-error.so.0.33.1
7fe7ed667000 r-xp 00005000 00:44 16253859      88   88    0         0         88         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libgpg-error.so.0.33.1
7fe7ed67d000 r--p 0001b000 00:44 16253859      44    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libgpg-error.so.0.33.1
7fe7ed688000 r--p 00026000 00:44 16253859       4    4    0         0          4         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libgpg-error.so.0.33.1
7fe7ed689000 r--p 00026000 00:44 16253859       4    4    0         0          4         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libgpg-error.so.0.33.1
7fe7ed68a000 ---p 00000000 00:00 0              8    8    1         1          8         0   0        0               0             0             0               0    0       0      0           0             0
7fe7ed68c000 r--p 00000000 00:44 16253855      44    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libgmp.so.10.4.1
7fe7ed697000 r-xp 0000b000 00:44 16253855     372    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libgmp.so.10.4.1
7fe7ed6f4000 r--p 00068000 00:44 16253855      92    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libgmp.so.10.4.1
7fe7ed70b000 r--p 0007f000 00:44 16253855       4    4    0         0          4         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libgmp.so.10.4.1
7fe7ed70c000 r--p 00080000 00:44 16253855       4    4    0         0          4         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libgmp.so.10.4.1
7fe7ed70d000 r--p 00000000 00:44 16253861      36    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libhogweed.so.6.6
7fe7ed716000 r--p 00009000 00:44 16253861      76    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libhogweed.so.6.6
7fe7ed729000 r--p 0001c000 00:44 16253861     168    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libhogweed.so.6.6
7fe7ed753000 r--p 00045000 00:44 16253861       8    8    1         1          8         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libhogweed.so.6.6
7fe7ed755000 r--p 00047000 00:44 16253861       4    4    0         0          4         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libhogweed.so.6.6
7fe7ed756000 r--p 00000000 00:44 16253876      52    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libnettle.so.8.6
7fe7ed763000 r-xp 0000d000 00:44 16253876     156    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libnettle.so.8.6
7fe7ed78a000 r--p 00034000 00:44 16253876      92    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libnettle.so.8.6
7fe7ed7a1000 r--p 0004b000 00:44 16253876       8    8    1         1          8         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libnettle.so.8.6
7fe7ed7a3000 rw-p 0004d000 00:44 16253876       4    4    0         0          4         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libnettle.so.8.6
7fe7ed7a4000 r--p 00000000 00:44 16253912      12    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libtasn1.so.6.6.3
7fe7ed7a7000 r-xp 00003000 00:44 16253912      48    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libtasn1.so.6.6.3
7fe7ed7b3000 r--p 0000f000 00:44 16253912      16    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libtasn1.so.6.6.3
7fe7ed7b7000 r--p 00013000 00:44 16253912       4    4    0         0          4         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libtasn1.so.6.6.3
7fe7ed7b8000 rw-p 00014000 00:44 16253912       4    4    0         0          4         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libtasn1.so.6.6.3
7fe7ed7b9000 r--p 00000000 00:44 16253921      72    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libunistring.so.2.2.0
7fe7ed7cb000 r-xp 00012000 00:44 16253921     256    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libunistring.so.2.2.0
7fe7ed80b000 r--p 00052000 00:44 16253921    1404    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libunistring.so.2.2.0
7fe7ed96a000 r--p 001b1000 00:44 16253921      16   16    2         2          0        16   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libunistring.so.2.2.0
7fe7ed96e000 rw-p 001b5000 00:44 16253921       4    4    0         0          4         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libunistring.so.2.2.0
7fe7ed96f000 r--p 00000000 00:44 16253863       8    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libidn2.so.0.3.8
7fe7ed971000 r-xp 00002000 00:44 16253863      28    0    0         0          0         0   0        0               0             0             0               0    0       0      0           0             0 /usr/lib/x86_64-linux-gnu/libidn2.so.0.3.8
...
```

Summarizing smaps for PostgreSQL Connection

credativ

- GO program - parse & pivot the /proc/PID/smaps file, show memory usage by path
- Summary for new connection - 42 different /usr/lib/x86_64-linux-gnu/ libraries
- And many small regions without paths -> summarized together as [anonymous]
- Shared buffers are mapped as "/dev/zero (deleted)" - see next slide

```
## output of top command
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 190747 postgres  20   0 8701512  20248  16876 S   0.0   0.1   0:00.00 postgres: postgres postgres 172.18.0.1(40278) idle

## script output - smaps
```

Path	Size	Rss	Pss	Pss_Dirty	Shr_Clean	Shr_Dirty	Prv_Clean	Prv_Dirty	Swap	SwapPss	Cnt
/usr/lib/postgresql/18/bin/postgres	9296	4140	1156	75	3792	168	128	52	0	0	5
[anonymous]	1708	660	554	554	0	120	0	540	0	0	21
[heap]	1440	1132	821	821	0	368	0	764	0	0	2
/dev/shm/PostgreSQL.1436672634	1024	132	130	130	0	4	0	128	0	0	1
/dev/shm/PostgreSQL.3104938386	112	4	1	1	0	4	0	0	0	0	1
/dev/zero (deleted)	8624208	10352	5070	5070	0	9780	0	572	0	0	1
/usr/lib/postgresql/18/lib/auto_explain.so	20	8	0	0	0	8	0	0	0	0	5
/usr/lib/postgresql/18/lib/pg_stat_statements.so	44	8	0	0	0	8	0	0	0	0	5
/usr/lib/locale/locale-archive	2980	60	19	0	60	0	0	0	0	0	1
/usr/lib/x86_64-linux-gnu/libffi.so.8.1.2	48	8	0	0	0	8	0	0	0	0	5
/usr/lib/x86_64-linux-gnu/libgpg-error.so.0.33.1	160	8	0	0	0	8	0	0	0	0	5
/usr/lib/x86_64-linux-gnu/libgmp.so.10.4.1	516	8	0	0	0	8	0	0	0	0	5
...											
/SYSV00ce5741 (deleted)	4	0	0	0	0	0	0	0	0	0	1
/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2	208	80	18	9	64	8	0	8	0	0	5
[stack]	132	36	27	27	0	12	0	24	0	0	1
[vvar]	16	0	0	0	0	0	0	0	0	0	1
[vdso]	8	4	0	0	4	0	0	0	0	0	1
Total	8701512	20380	8553	6841	6356	11760	164	2100	0	0	251

Why "/dev/zero (deleted)" ?

credativ

- Why shared buffers are mapped as `/dev/zero (deleted)` " in `smaps` output?
  - PostgreSQL requests "shared anonymous memory" from OS
  - Using `mmap()` system call with `MAP_ANONYMOUS | MAP_SHARED` flags
  - I.e. "shared memory with anonymous mapping (not backed by any file)" is requested
  - Described in PG code as "anonymous mmap()ed shared memory segment"
  - Hence PG setting shared_memory_type = 'mmap'
- How Linux implements this internally?
  - Linux kernel internally instantiates a synthetic file object within `tmpfs`
  - Kernel source code names it "dev/zero" - legacy from older implementations
  - This internal backing file serves only as a handle for memory management
  - It is not linked to Virtual File System (VFS) directory tree
  - Therefore it shows up as "(deleted)" in `smaps` output
- Hence shared buffers are mapped as `/dev/zero (deleted)`

Connection Memory Usage After Running Query

- Heavy aggregations over the table with JSONB data, not fitting into memory
- Memory 32 GB, table 38 GB, shared_buffers 8 GB, work_mem 64 MB, max_parallel_workers_per_gather = 0
- Huge numbers are only in the line mapping shared buffers - ”/dev/zero (deleted)”

```
## top command output after query run
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 190747 postgres  20   0 8701848   8.2g   8.1g S   0.0  26.3   0:48.67 postgres: postgres postgres 172.18.0.1(40278) idle

## smaps numbers after query run
Path                                            Size        Rss        Pss  Pss_Dirty  Shr_Clean  Shr_Dirty  Prv_Clean  Prv_Dirty      Swap  SwapPss  Cnt
-------------------------------------------------------------------------------------------------------------------------------------------------------------
/usr/lib/postgresql/18/bin/postgres             9296       6508       3255         79       4176        164       2112         56         0        0    5
[anonymous]                                     1708        704        598        598          0        120          0        584         0        0   21
[heap]                                          1776       1516       1208       1208          0        364          0       1152         0        0    2
/dev/shm/                                       1136        148        143        143          0          8          0        140       980        0    2
/dev/zero (deleted)                          8624208    8532008    8443508    8443508          0     143376          0    8388632         0        0    1
/usr/lib/postgresql/18/lib/                       64         44         22          8         28          8          0          8         0        0   10
/usr/lib/locale/locale-archive                  2980         68         19          0         68          0          0          0         0        0    1
/usr/lib/x86_64-linux-gnu/                     60520       5020       1376        167       3556       1284        156         24         0        0  205
/SYSV00ce5741 (deleted)                            4          0          0          0          0          0          0          0         0        0    1
[stack]                                          132         44         44         44          0          0          0         44         0        0    1
[vvar]                                            16          0          0          0          0          0          0          0         0        0    1
[vdso]                                             8          4          0          0          4          0          0          0         0        0    1
-------------------------------------------------------------------------------------------------------------------------------------------------------------
Total                                        8701848    8546064    8450173    8445755       7832     145324       2268    8390640       980        0  251
```

How Much Memory Are Connections Really Using

credativ

- Back to original example with multiple sessions -> summary of data from smaps files
- With shared buffers numbers are horrible -> huge values as shown in ”top” command
- Without shared buffers -> connections are actually using very small amounts of memory

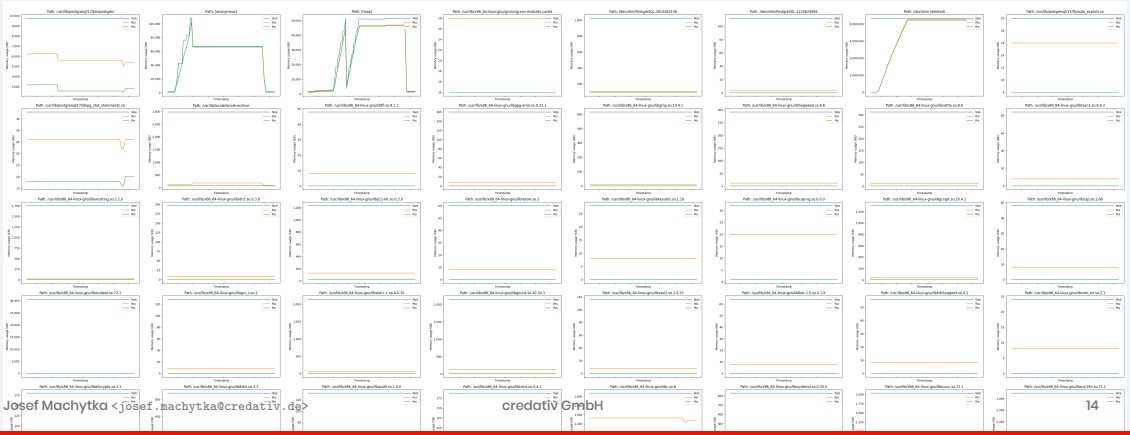
```
## top command
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
190747 postgres  20   0 8701868   8.1g   8.1g S   0.0  26.2   0:48.69 postgres: postgres postgres 172.18.0.1(40278) idle
206119 postgres  20   0 8702808   4.7g   4.7g S   0.0  15.2   0:21.81 postgres: postgres postgres 172.18.0.1(44010) idle
219065 postgres  20   0 8802384   8.2g   8.1g S   0.0  26.6   2:56.64 postgres: postgres postgres 172.18.0.1(52912) idle
228832 postgres  20   0 8709912   3.4g   3.4g S   0.0  11.1   0:11.10 postgres: postgres postgres 172.18.0.1(60090) idle
230390 postgres  20   0 8701340   8.1g   8.1g S   0.0  26.3   0:51.89 postgres: postgres postgres 172.18.0.1(44802) idle

## smaps summaries with /dev/zero
Path                                           Size        Rss        Pss  Pss_Dirty  Shr_Clean  Shr_Dirty  Prv_Clean  Prv_Dirty      Swap  SwapPss  Cnt
-------------------------------------------------------------------------------------------------------------------------------------------------------
Total for /proc/190747/smaps                8701868    8529172    2196188    2195833       2960    8525332          0        880     61784     1116  256
Total for /proc/206119/smaps                8702808    4928096    1119095    1118712       3120    4924968          0          8     63996     2112  258
Total for /proc/219065/smaps                8802384    8635640    2296848    2295726       5472    8531892         12      98264     64896     4078  252
Total for /proc/228832/smaps                8709912    3609444     798269     795595       8660    3590600         60      10124     60936      100  252
Total for /proc/230390/smaps                8701340    8542712    2202431    2199069       8660    8531564        748       1740     60768       99  251
-------------------------------------------------------------------------------------------------------------------------------------------------------
                                           43618312   34285064    8611831    8613495      34872   34193356        820      19916    312480    17405 1279
## smaps summaries without /dev/zero
Path                                           Size        Rss        Pss  Pss_Dirty  Shr_Clean  Shr_Dirty  Prv_Clean  Prv_Dirty      Swap  SwapPss  Cnt
-------------------------------------------------------------------------------------------------------------------------------------------------------
Total for /proc/190747/smaps                  77660       4416       1291        936       2960        576          0        880      3508     1116  255
Total for /proc/206119/smaps                  78600       3708        448         65       3120        580          0          8      5720     2112  257
Total for /proc/219065/smaps                 178176     104316      99427      98305       5472        584         12      98248      6620     4078  251
Total for /proc/228832/smaps                  85704      19420      12853      10179       8660        576         60      10124      2660      100  251
Total for /proc/230390/smaps                  77132      11716       5143       1781       8660        584        748       1724      2492       99  250
-------------------------------------------------------------------------------------------------------------------------------------------------------
                                             499272     169576     118162     110266      34872       2900        820      19984     18300    17405 1274
```

How to measure memory usage of a query?

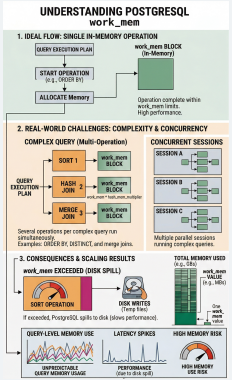
credativ

- GO program scrapes & stores smaps snapshots during query run
- Another GO program parses & pivots smaps data, Python script plots the data
- Only 3 paths are really interesting - heap, anonymous and shared memory



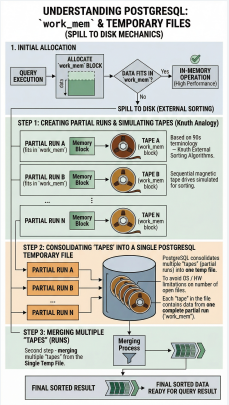
What Is That work_mem Anyway

- Base maximum memory for a query operation
- Applies for operations like sort or hash table
- Sort - ORDER BY, DISTINCT, and merge joins
- Hash-based operations use hash_mem_multiplier
- When exceeded, PostgreSQL writes to disk
- Complex query - several such operations at the same time
- Several sessions can run complex queries in parallel
- Total memory used can be many times the value of work_mem



Work_mem and Temporary Files

- If data fits into work_mem -> in-memory operation
- If not -> spilling to disk into ”temporary tapes”
  - Based on terminology of 90s - Knuth algorithms
  - Sequential magnetic tape drives used for sorting
- PG creates a ”temporary file” for multiple ”tapes”
  - To avoid OS / HW limitations for number of files
  - Each ”tape” contains data from one partial run (work_mem)
  - src/backend/utils/sort/logtape.c implements space recycling
  - Multiple merge passes could otherwise blow space usage
  - temp_file_limit - limits total size of all temp files per process
- Second step - merging multiple ”tapes” into one



Temporary Files – Details

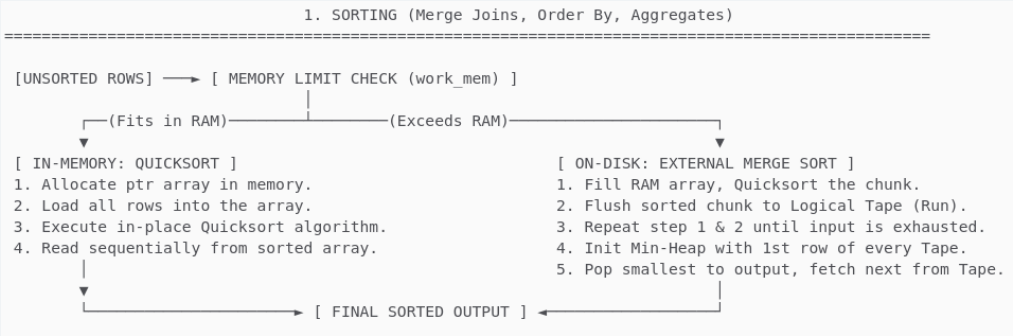
credativ

- `src/backend/storage/file/buffile.c` - abstraction for temporary files
  - Encapsulates multiple OS files into one logical file
  - To avoid OS / HW limitations for number / size of files
  - Temp files can be shared between processes for parallel execution
  - Even files which must survive across transactions

- Operations with temporary files visible as wait events
  - BuffileRead - Waiting for a read from a buffered file
  - BuffileTruncate - Waiting for a buffered file to be truncated
  - BuffileWrite - Waiting for a write to a buffered file

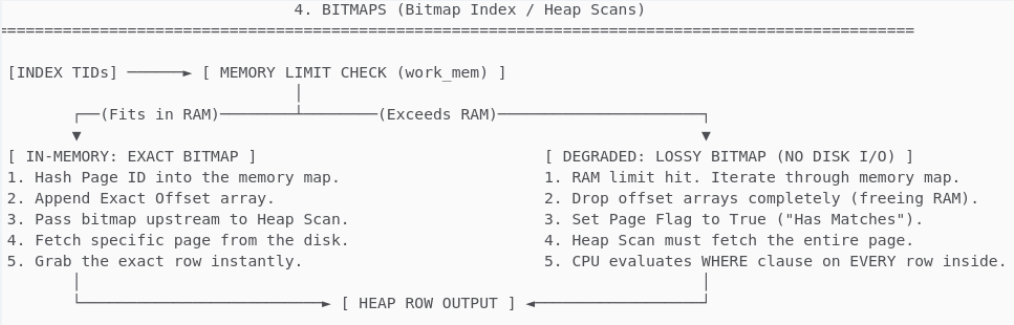
Sort Operations can be spilled to disk

- `src/backend/utils/sort/tuplesort.c` or `tuplestore.c`
- Either CPU-bound array sort fully in memory or I/O-heavy external merge sort
- EXPLAIN ANALYZE: Sort Method: external merge Disk: 123456kB



BitMap Index / Heap Scan - Only in Memory

- Exact scan only if it fits into memory - array of pointers to heap tuples
- If memory is exceeded, array is dropped, lossy scan examines whole pages
- EXPLAIN ANALYZE: Bitmap Heap Scan - Heap Blocks: exact=50 lossy=450 (reporting fixed in PG18)

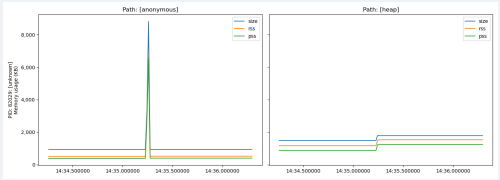


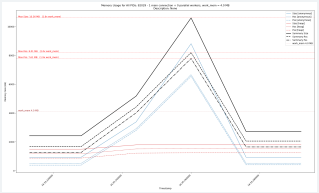
ORDER BY on 100,000 data points

- Very naive and simple sorting query, fitting into work_mem = 4MB
- Other operations show the same memory usage pattern with small data sets
- Higher work_mem values show exactly the same query plan and memory usage
- This is what developers see with small testing data sets - quick operations, no memory issues

```
-- RAM = 32 GB - ; JIT = off; shared_buffers = '8GB'; work_mem = '4MB';
SELECT * FROM generate_series(1, 100000) AS t(val) ORDER BY val;

 Sort  (cost=9304.82..9554.82 rows=100000 width=4) (actual time=10.409..12.685 rows=100000.00 loops=1)
   ...
   Sort Method: quicksort  Memory: 3073kB
   ...
 Planning:
   Memory: used=9kB  allocated=16kB
```





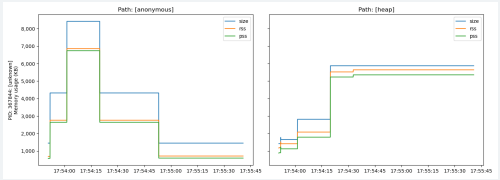
2x work_mem

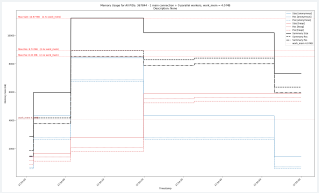
ORDER BY on 100,000,000 data points

- Bigger data set spills to disk
- Very small work_mem + big data -> heap memory often bloated after query (here 1.5x work_mem)

```
-- RAM = 32 GB - JIT = off; shared_buffers = '8GB'; work_mem = '4MB';
SELECT * FROM generate_series(1, 100000000) AS t(val) ORDER BY val DESC;
```

```
Sort  (cost=18389274.88..18639274.88 rows=100000000 width=4) (actual time=28096.831..32774.182 rows=100000000.00 loops=1)
  ...
  Sort Method: external merge  Disk: 1174280kB
  Buffers: shared hit=3, temp read=611239 written=611811
  I/O Timings: temp read=764.458 write=1084.815
  ->  Function Scan on pg_catalog.generate_series t  (cost=0.00..1000000.00 rows=100000000 width=4) (actual time=5771.300..10295.004 rows=100000000.00 loops=1)
        ...
        I/O Timings: temp read=187.325 write=470.169
Planning:
  Memory: used=9kB  allocated=16kB
```

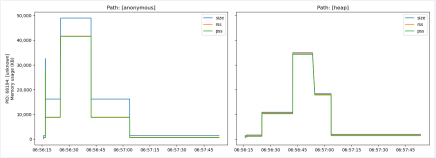


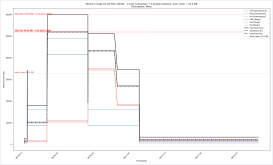


2.2x work_mem

ORDER BY on 100,000,000 data points - different work_mem

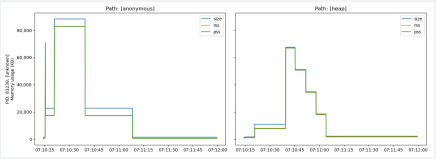
- Higher work_mem eliminates heap bloat due to different allocation mechanism
- work_mem = 32MB

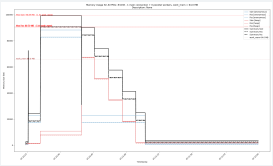




1.6x work_mem

- work_mem = 64MB

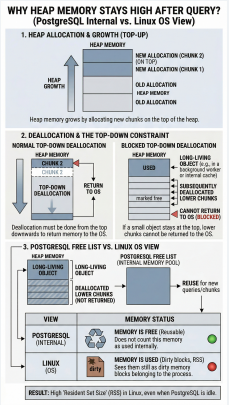




1.4x work_mem

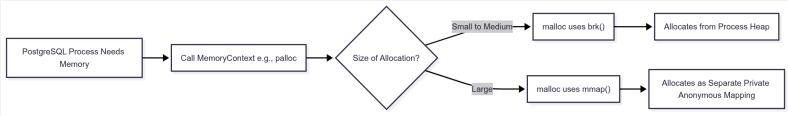
Why does Heap memory stay high after a query?

- Heap memory grows by allocating new chunks on the top of the heap
- Deallocation must be done from the top of the heap downwards
- A query can create some long-living small object on top
- Subsequently deallocated lower chunks are only placed on the free list
- PostgreSQL internally does not count this memory as used
- Linux sees them still as dirty memory blocks belonging to the process



Why does a small work_mem use more heap?

- PostgreSQL uses palloc / MemoryContext to obtain large blocks of memory
- Sizes the executor / aggregation memory context based on work_mem
- Underlying glibc malloc allocation strategy depends on size of allocation
  - Small to medium blocks allocated via brk() from process heap
  - Large blocks allocated via mmap() as separate private anonymous mappings
- Therefore we see 2 distinct memory usage patterns
  - small blocks -> brk() -> heap memory
  - large blocks -> mmap() -> separate anonymous mappings

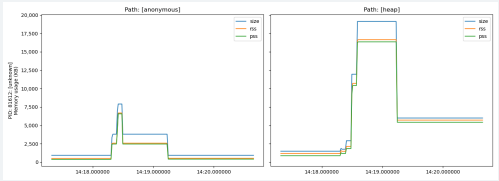


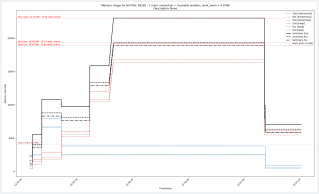
Merge Join & Materialize – 1,000,000 data points

- work_mem = 4MB, hash_mem_multiplier almost no effect: 1 -> 4.3x, 2 & higher -> 4.7x work_mem
- The same memory usage also for 100,000,000 data points, similarly no effect of hash_mem_multiplier

```
-- RAM = 32 GB - JIT = off; shared_buffers = '8GB'; work_mem = '4MB';
SELECT * FROM generate_series(1, 1000000) AS a(id) JOIN generate_series(1, 1000000) AS b(id) USING (id);
```

```
Merge Join  (cost=246664.69..75254164.69 rows=500000000000 width=4) (actual time=368.631..613.185 rows=1000000.00 loops=1)
  ------
  I/O Timings: temp read=6.391 write=14.626
  ->  Sort  (cost=123322.35..125822.35 rows=1000000 width=4) (actual time=130.730..179.787 rows=1000000.00 loops=1)
        ------
        Sort Method: external merge  Disk: 11768kB
        Buffers: shared hit=4, temp read=2192 written=2192
        I/O Timings: temp read=6.391 write=14.626
        ->  Function Scan on pg_catalog.generate_series a  (cost=0.00..10000.00 rows=1000000 width=4) (actual time=47.359..94.206 rows=1000000.00 loops=1)
              ------
              Buffers: temp read=1709 written=1709
              I/O Timings: temp read=4.779 write=4.414
  ->  Materialize  (cost=123322.35..125822.35 rows=1000000 width=4) (actual time=137.890..293.575 rows=1000000.00 loops=1)
        ------
        Storage: Memory  Maximum Storage: 47kB
        Buffers: temp read=1709 written=1709
        I/O Timings: temp read=4.138 write=7.088
        ->  Sort  (cost=123322.35..125822.35 rows=1000000 width=4) (actual time=137.881..193.384 rows=1000000.00 loops=1)
              ------
              Sort Method: external merge  Disk: 11768kB
              Buffers: temp read=1709 written=1709
              I/O Timings: temp read=4.138 write=7.088
              ->  Function Scan on pg_catalog.generate_series b  (cost=0.00..10000.00 rows=1000000 width=4) (actual time=44.462..83.998 rows=1000000.00 loops=1)
                    ------
                    Buffers: temp read=1709 written=1709
                    I/O Timings: temp read=4.073 write=3.933
Planning:
  Memory: used=17kB  allocated=24kB
```





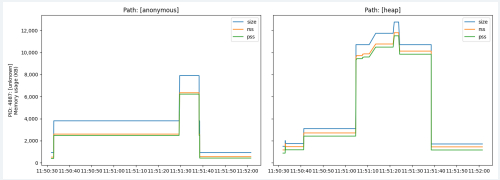
4.7x work_mem

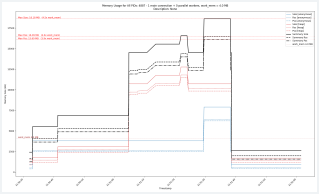
Hash Aggregation with array_agg() – 100,000,000 data points

- work_mem = 4MB, GROUP BY with array_agg() function
- Here I have seen the influence of hash_mem_multiplier -> here = 2 (default)
- Spilled 2.6 GB to disk, used 4x work_mem in connection, bloat 0.5x work_mem

```
SELECT (val/100)::int as grp, array_agg(val) FROM generate_series(1, 100000000) t(val) GROUP BY grp;
```

```
HashAggregate  (cost=1750000.00..1750003.00 rows=200 width=36) (actual time=325027.127..350440.860 rows=1000001.00 loops=1)
  ...
  Batches: 261  Memory Usage: 8361kB  Disk Usage: 2779568kB
  Buffers: temp read=847165 written=1107020
  I/O Timings: temp read=2476.554 write=39221.438
  ->  Function Scan on pg_catalog.generate_series t  (cost=0.00..1250000.00 rows=100000000 width=8) (actual time=9034.350..145341.188 rows=100000000.00 loops=1)
        ...
        Buffers: temp read=170899 written=170899
        I/O Timings: temp read=561.884 write=1984.829
Planning:
  Buffers: shared hit=22
  Memory: used=13kB  allocated=16kB
```





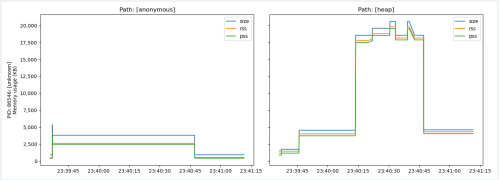
4x work_mem

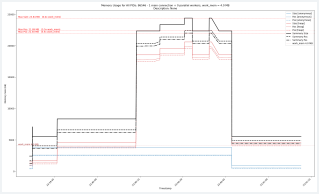
Hash Aggregation with array_agg() – 100,000,000 data points

- work_mem = 4MB, GROUP BY with array_agg() function
- Higher hash_mem_multiplier = 4
- Spilled 2.6 GB to disk, used 5.5x work_mem in connection, bloat 1.1x work_mem

```
SELECT (val/100)::int as grp, array_agg(val) FROM generate_series(1, 100000000) t(val) GROUP BY grp;
```

```
HashAggregate  (cost=1750000.00..1750003.00 rows=200 width=36) (actual time=295092.652..320497.389 rows=1000001.00 loops=1)
  ...
  Batches: 133  Memory Usage: 16425kB  Disk Usage: 2735976kB
  Buffers: temp read=837451 written=1092253
  I/O Timings: temp read=2468.626 write=7624.314
  ->  Function Scan on pg_catalog.generate_series t  (cost=0.00..1250000.00 rows=100000000 width=8) (actual time=8434.410..145393.926 rows=100000000.00 loops=1)
        ...
        Buffers: temp read=170899 written=170899
        I/O Timings: temp read=587.218 write=1215.390
Planning:
  Buffers: shared hit=22
  Memory: used=13kB  allocated=16kB
```





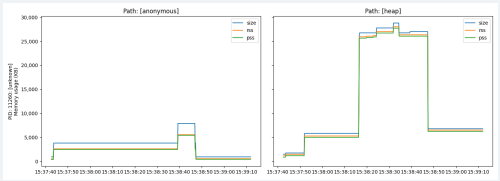
5.5x work_mem

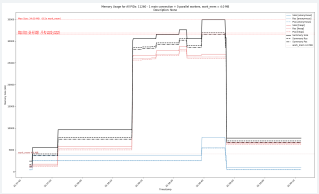
Hash Aggregation with array_agg() – 100,000,000 data points

- work_mem = 4MB, GROUP BY with array_agg() function
- Higher hash_mem_multiplier = 6
- Spilled 2.6 GB to disk, used 7.8x work_mem in connection, bloat 2x work_mem

```
SELECT (val/100)::int as grp, array_agg(val) FROM generate_series(1, 100000000) t(val) GROUP BY grp;
```

```
HashAggregate  (cost=1750000.00..1750003.00 rows=200 width=36) (actual time=292963.547..317835.434 rows=1000001.00 loops=1)
  ...
  Batches: 133  Memory Usage: 24617kB  Disk Usage: 2733000kB
  Buffers: temp read=828088 written=1082605
  I/O Timings: temp read=2415.069 write=6982.424
  ->  Function Scan on pg_catalog.generate_series t  (cost=0.00..1250000.00 rows=100000000 width=8) (actual time=8283.160..144507.248 rows=100000000.00 loops=1)
        ...
        Buffers: temp read=170899 written=170899
        I/O Timings: temp read=582.129 write=1268.022
Planning:
  Buffers: shared hit=22
  Memory: used=13kB  allocated=16kB
```





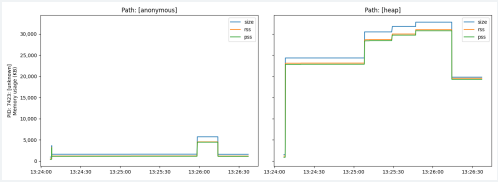
7.8x work_mem

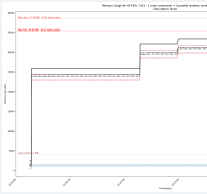
JIT ON & Hash Aggregation with array_agg()

- work_mem = 4MB, hash_mem_multiplier = 2, GROUP BY with array_agg() function, JIT ON
- Without JIT - Spilled 2.6 GB to disk, used 4x work_mem in connection, bloat 0.5x work_mem
- With JIT ON - Spilled 2.6 GB to disk, used 8.7x work_mem in connection, bloat 5x work_mem

```
SELECT (val/100)::int as grp, array_agg(val) FROM generate_series(1, 100000000) t(val) GROUP BY grp;

 HashAggregate  (cost=1750000.00..1750003.00 rows=200 width=36) (actual time=316653.347..339198.433 rows=1000001.00 loops=1)
   ...
   Batches: 261  Memory Usage: 8369kB  Disk Usage: 2779568kB
   ...
Planning:
  Buffers: shared hit=22
  Memory: used=13kB  allocated=16kB
JIT:
  Functions: 14
  Options: Inlining true, Optimization true, Expressions true, Deforming true
  Timing: Generation 0.817 ms (Deform 0.172 ms), Inlining 66.045 ms, Optimization 43.470 ms, Emission 25.082 ms, Total 135.414 ms
```

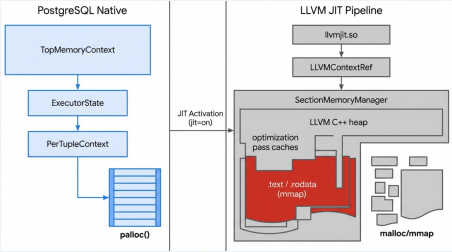




8.7x work_mem

Why is JIT so memory hungry?

- JIT is great for CPU-bound analytical workloads, but eats memory
- Relies on LLVM (Low Level Virtual Machine) compiler infrastructure
- Dynamically loaded llvmjit.so library relies on standard OS allocators



Summary

credativ

- Small work_mem setting does not guarantee small memory usage
- RSS values of PostgreSQL connections are misleading
  - RSS of each connection includes some part of, or even the full size of, shared_buffers
  - Real memory usage of empty connection is small, usually a few dozen MBs
- Anonymous and heap memory together during a query run use more than a single work_mem
  - Heap memory can stay bloated after query finishes
  - Small work_mem + big data = many temp files + bloated heap memory
  - Internally PostgreSQL sees this memory as deallocated, on free list
- JIT = ON can blow memory usage into unexpected numbers
  - 8x work_mem during a query run,
  - Bloated heap 4x-6x work_mem after a query finishes
- Regular restart of connections is highly recommended

Thank you for your attention!

credativ

All my slides

Recorded talks