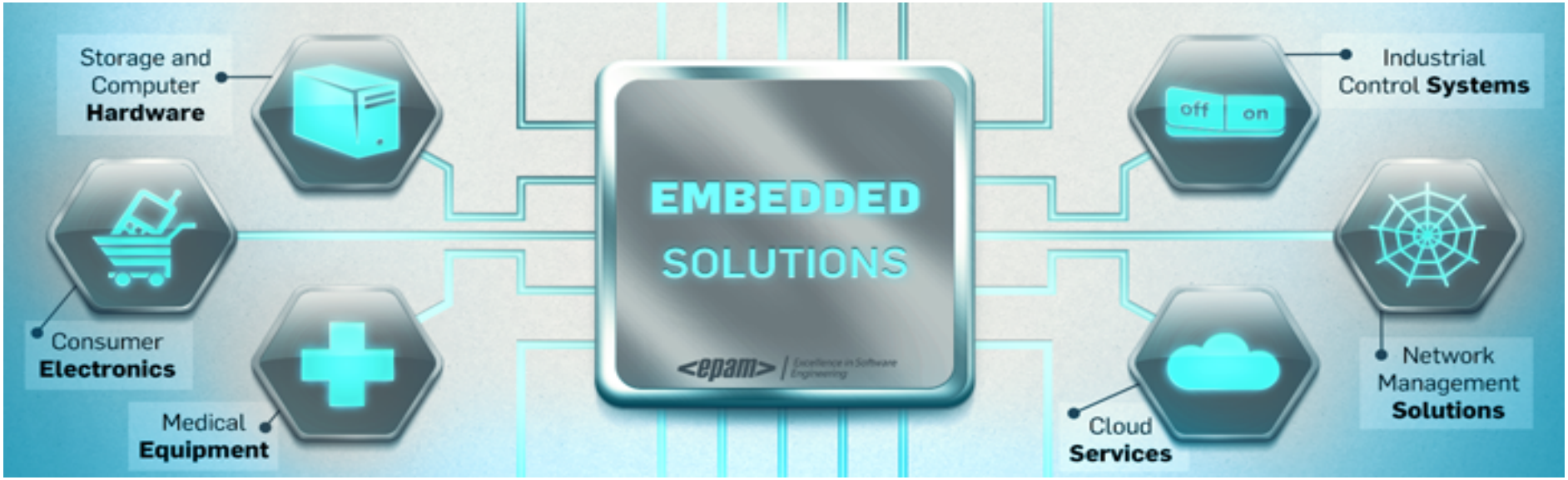
<epam> Excellence in Software Engineering

http://www.epam.com/

#>LLPD



http://www.epam.com/solutions/embedded.html

Ivan Matylitski

https://github.com/buffovich

Flexible user-level implementation of SLAB-like free list

Flexible user-level implementation of SLAB-like free list

Basic ideas about SLAB

- History
- Goal
- Structure

My own cute SLAB

- Limitations of existing implementations
- Features
- Architecture
- Involved technologies and science
- My own inventions
  -

History of SLAB

solaris™

SLAB structure was invented by Jeff Bonwick and introduced in the Solaris 2.4 kernel.

History of SLAB

He needed faster algorithm for massive allocation-deallocation of structures of the same type:

- thread descriptors
- file descriptions
- etc

… instead of regular “best-fit” or “first-fit” allocator.

Jeff noticed that we can predict quantity and size of objects which will be needed in future in particular cases (thread structures are needed always)

Goal of SLAB

The vast majority of programs allocate, initialize, finalize and deallocate a lot of objects of the same type during they execution.

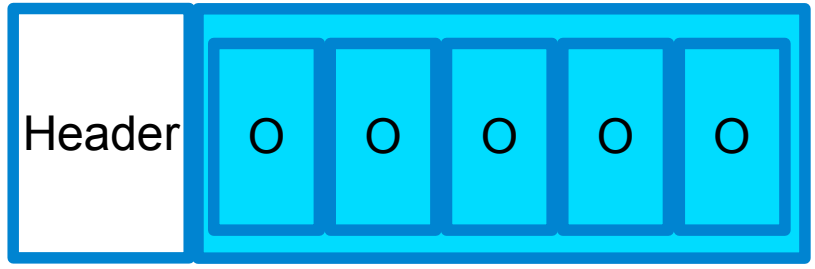
The idea of SLAB (as well as free list) is why don't recycle existing objects to avoid allocation and initialization overheads?

SLAB structure and mechanics

All allocations and object initialization are performed in “chunks” or “slabs” with reasonable amount of objects inside.

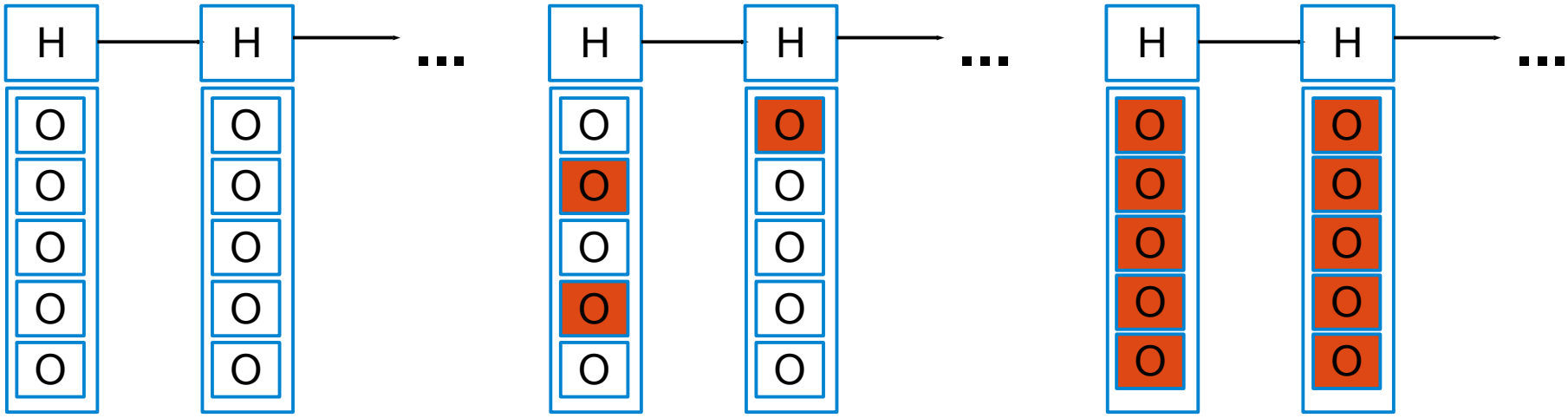
Probably, that's where “SLAB” came from.

Chunk consists of header (objects number, next-link, free map, etc) and array of instances. Header may have optional padding at the very end of header structure.



SLAB structure and mechanics

SLAB has three different lists:



1) *Empty list* where chunks don't have any occupied instance.

1) *Partial list* where chunks have both occupied and free instances.

1) *Full list* where chunks don't have any instances for allocation.

SLAB structure and mechanics

Linux kernel as one of the popular SLAB users:

Kernel API:

```
struct kmem_cache *
kmem_cache_create(const char *name, size_t
size, size_t align, unsigned long flags,
void (*ctor)(void *));

void * kmem_cache_alloc(struct kmem_cache
*cachep, gfp_t flags);

kmem_cache_free(task_struct_cachep, tsk);

kmem_cache_destroy(task_struct_cachep);
```

My own SLAB implementation

Limitations of existing implementations:

1) kernel SLAB – code isn't usable in userspace because of kernel low level style and system tricks;

2) pool implementation in Apache Runtime Library – tied to **libapr**, absence of flexibility and “configurability”;

3) pool implementation in Boost – tied to C++ and Boost; isn't directly applicable in some important cases because of the lack of “configurability”;

5) several implementations I met at GitHub during investigation – very simple and not thread-aware.

My own SLAB implementation

Killing feature is...

… that user can choose almost everything:

- particular back-end for memory allocation at compilation stage (ptmalloc, je_malloc, tcmalloc is currently available);
- particular scheme of behavior in multi-threaded environment at run-time:
  - local lists for each thread (disjoint access);
  - global, per-SLAB lists;
- scheme of protecting of lists from corruption under multi-threaded contention at run-time:
  - simple thread-unaware scheme;
  - coarse-grained locks for three lists;
  - fine-grained locks;
  - lock-less scheme.
- optional cache-coloring

My own SLAB implementation

Another points are...

- **Project is not attended to particular library/framework/environment; you can use it absolutely separately;**
- **Optional reference counting for block**
- **Constructors/destructors/”recyclers” support**

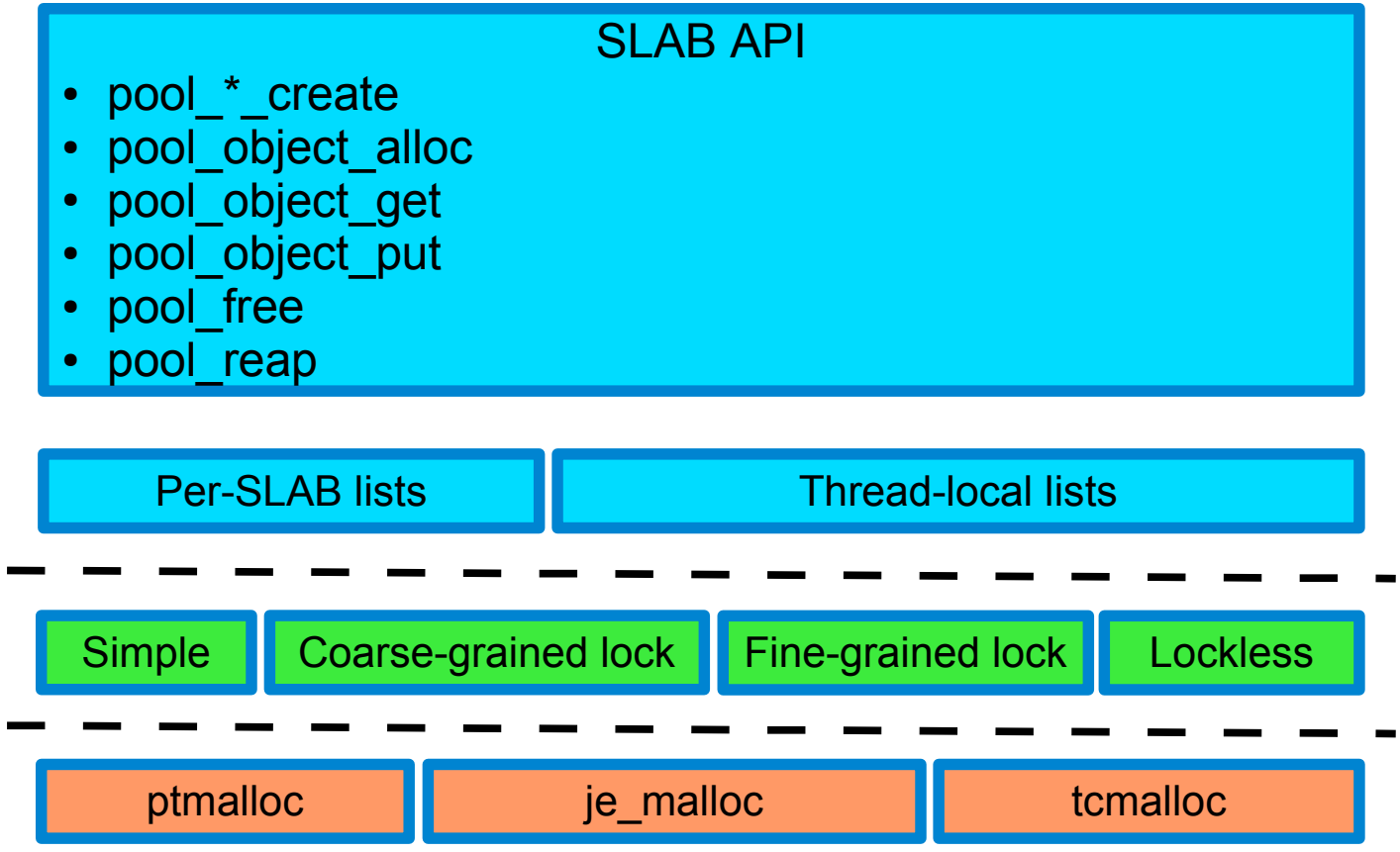
My own SLAB implementation

Outcome from just described features is...

- Applicable for the wide range of cases:
  - Low-memory devices;
  - Real-time applications;
  - Low-powered devices;
  - HA applications
- Provides tools with justified trade-offs – you may assemble what suits you in particular case.

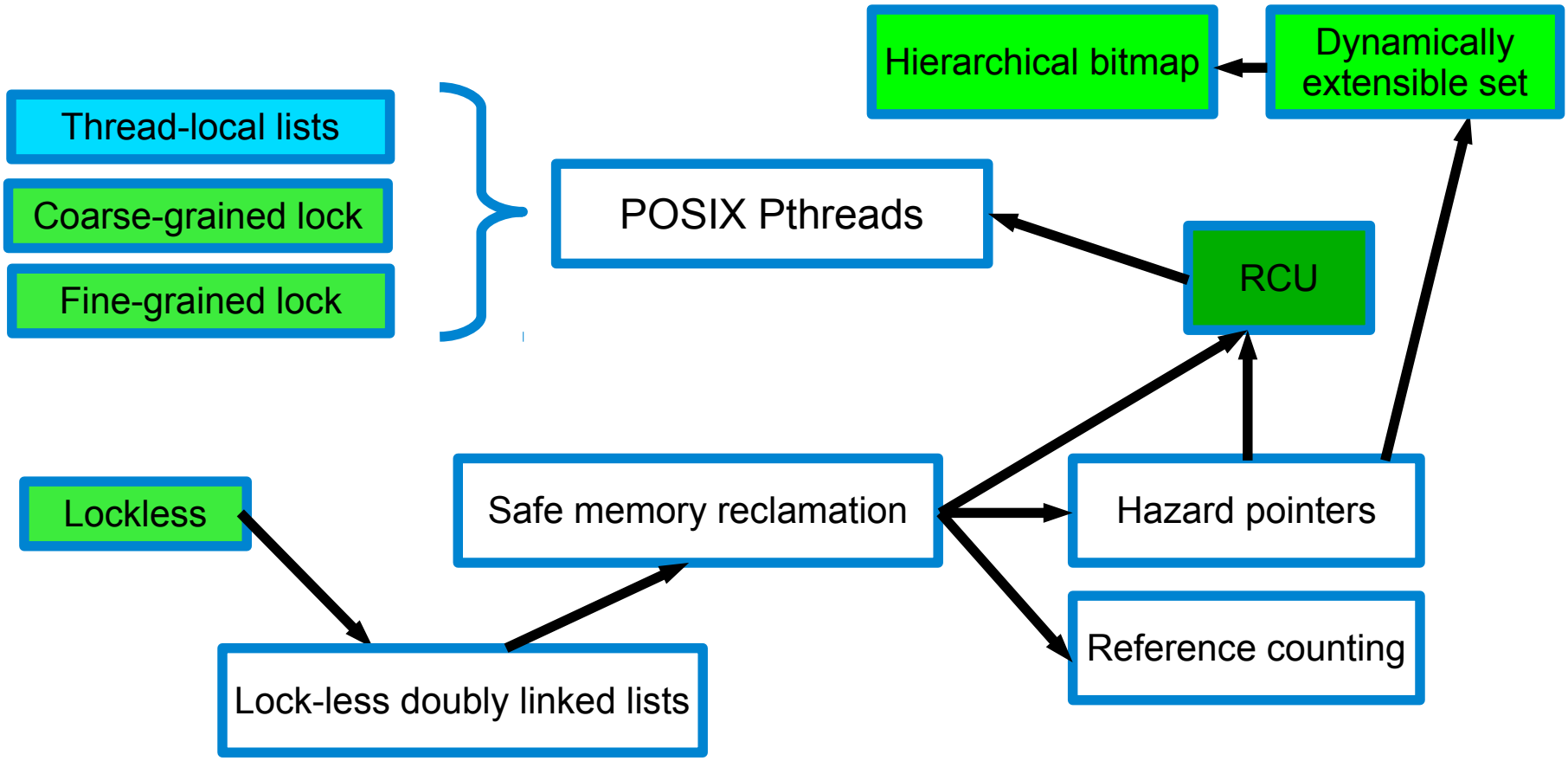
My own SLAB implementation

Architecture



My own SLAB implementation

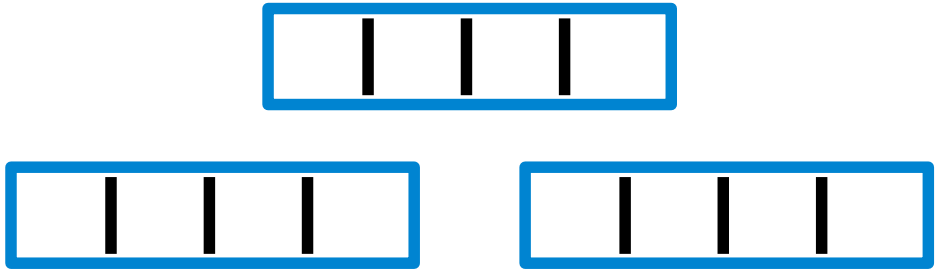
Technologies map



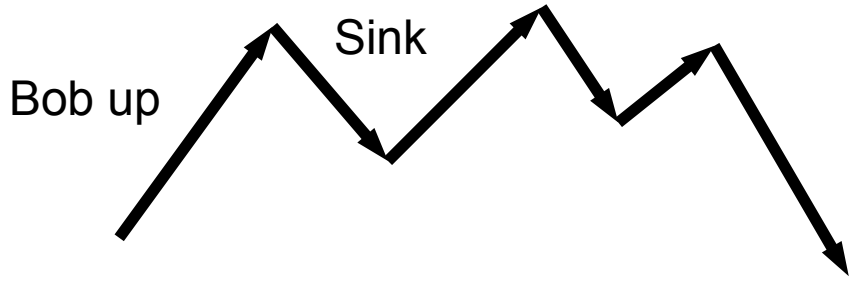
My own SLAB implementation

My own invention – hierarchical bitmap

Scheme:



Search algorithm:



Thanks to ...

Denis Pynkin (EPAM Systems)

Artem Sheremet (EPAM Systems)

Alexey Cheusov (Invention Machine)