Troubridge: Learning-Based Access-Frequency Prediction for Memory Allocation

Bokai Bi
Zi Qiao
Patrick Rui De Jing

Background

Increase in **in-memory computing** for big-data workloads

redis

apache Ignite™

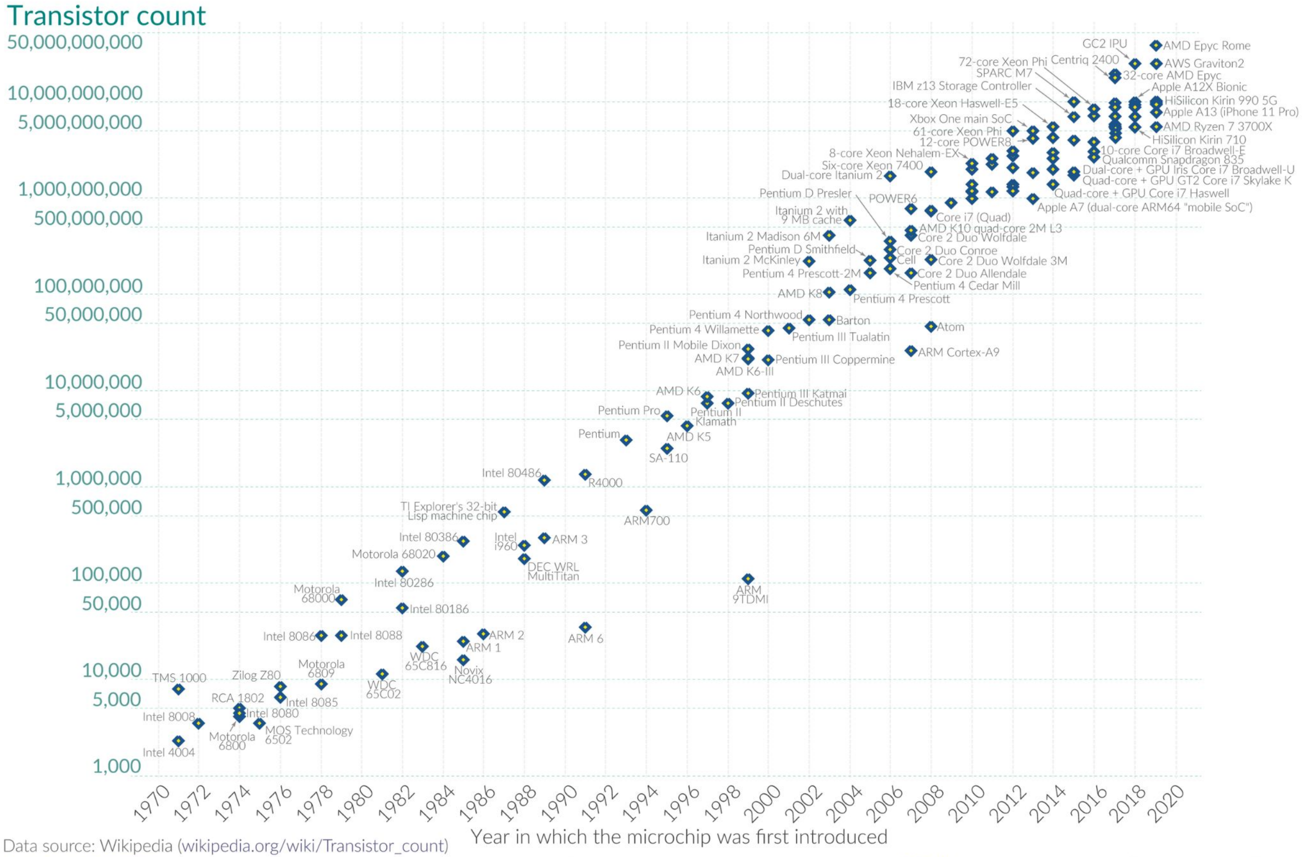
Google Bigtable

MEM GRAPH

Amazon Elasticache

Moore’s Law: The number of transistors on microchips doubles every two years

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

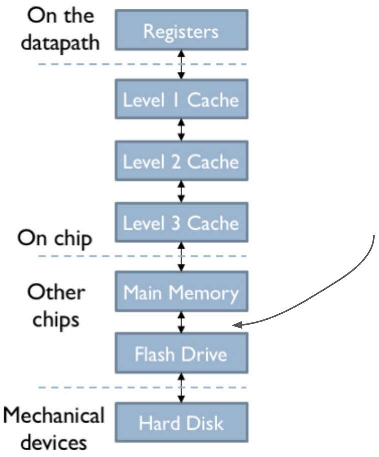
OurWorldinData.org – Research and data to make progress against the world’s largest problems.

Problem

1. Memory demand is higher than ever (in-memory computing)
2. Memory is not getting cheaper (end of Moore’s law)

Solution 1: Far Memory

1. Create a tier of memory called far memory



Far memory

Faster than flash

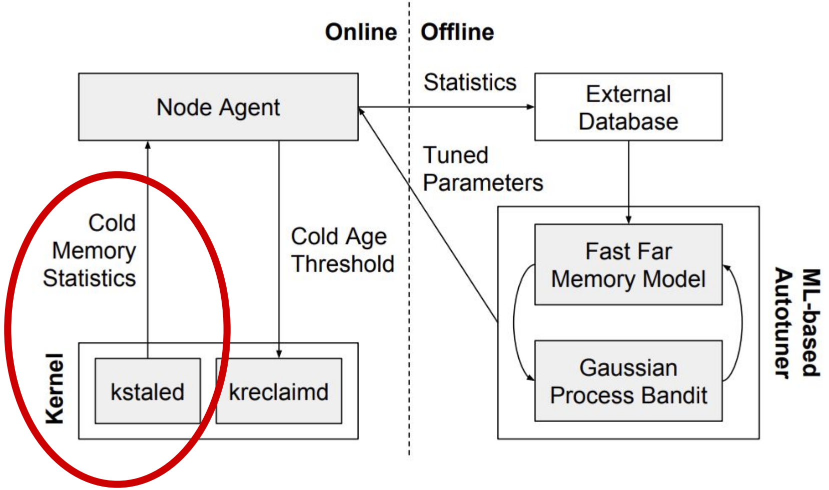
Cheaper than DRAM

2. Use far memory to store infrequently accessed (*cold*) memory
3. Perform the *same* jobs with *less* DRAM

Software-Defined Far Memory in Warehouse-Scale Computers

By Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, and Colin Raffe

Problem: Far memory systems rely on **kernel daemons** to collect statistics.

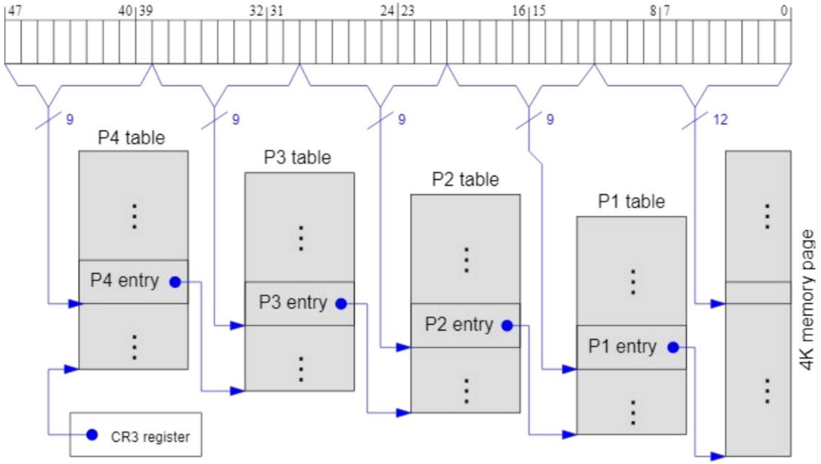


Can we identify cold memory *at allocation time*?

Solution 2: Hugepages

2 MB hugepages instead of 4KB pages

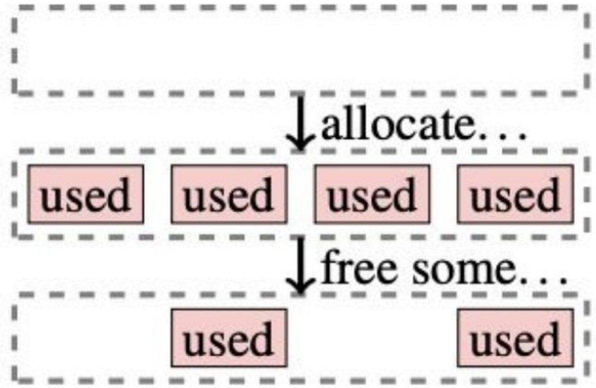
1. Reduce TLB cache misses
2. Decrease pagetable walks



Beyond malloc efficiency to fleet efficiency: a hugepage-aware memory allocator

By A.H. Hunter, Chris Kennelly, Paul Turner, Darryl Gove, Tipp Moseley, and Parthasarathy Ranganathan

Problem: Allocating many hugepages increases **fragmentation**



Optimally, we put *hot allocations* on hugepages and keep them segregated from *cold data.*

Problem

1. Memory demand is higher than ever (in-memory computing)
2. Memory is not getting cheaper (end of Moore’s law)

Solution(s)

1. Put cold data in **far memory**
2. Cluster hot allocations together in **hugepages**

We need to be sensitive to hotness!

Research Question

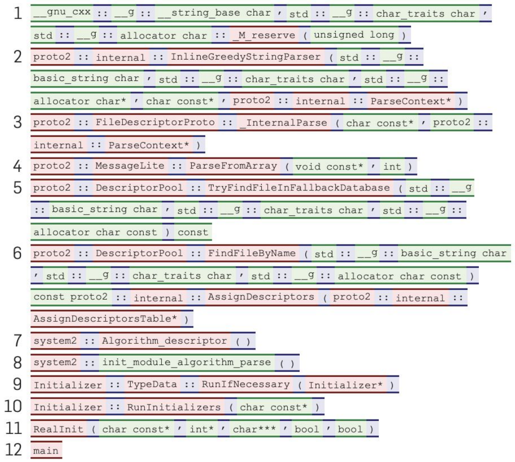
Research Question

Can we predict the hotness of an object *at allocation time* using the object size and stack trace?

Learning-based Memory Allocation for C++ Server Workloads

By Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, and Colin Raffe

[The model] predicts the lifetime of allocated objects using the stack trace...

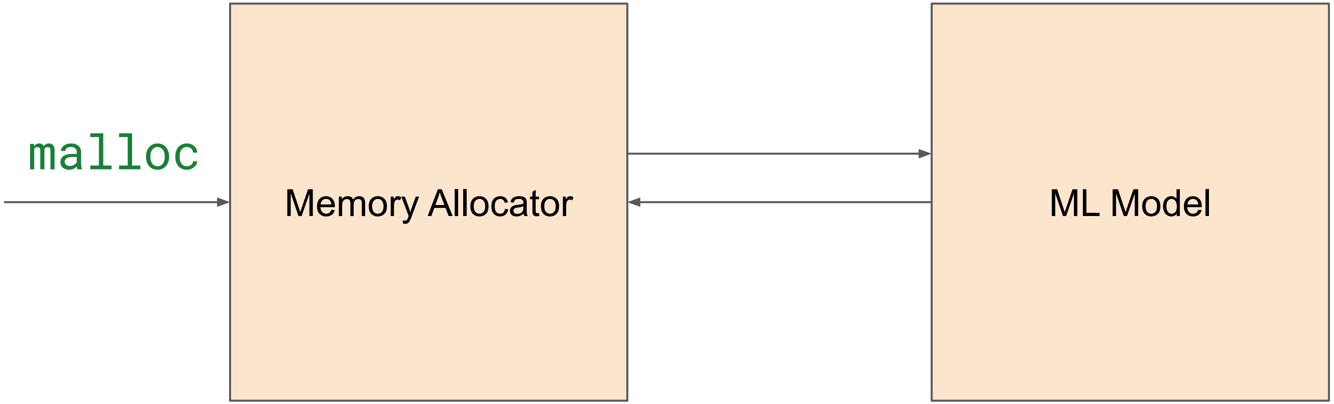


```
1  __gnu_cxx :: __g :: __string_base char , std :: __g :: char_traits char ,
   std :: __g :: allocator char :: _M_reserve ( unsigned long )
2  proto2 :: internal :: InlineGreedyStringParser ( std :: __g ::
   basic_string char , std :: __g :: char_traits char , std :: __g ::
   allocator char* , char const* , proto2 :: internal :: ParseContext* )
3  proto2 :: FileDescriptorProto :: _InternalParse ( char const* , proto2 ::
   internal :: ParseContext* )
4  proto2 :: MessageLite :: ParseFromArray ( void const* , int )
5  proto2 :: DescriptorPool :: TryFindFileInFallbackDatabase ( std :: __g
   :: basic_string char , std :: __g :: char_traits char , std :: __g ::
   allocator char const ) const
6  proto2 :: DescriptorPool :: FindFileByName ( std :: __g :: basic_string char
   , std :: __g :: char_traits char , std :: __g :: allocator char const )
   const proto2 :: internal :: AssignDescriptors ( proto2 :: internal ::
   AssignDescriptorsTable* )
7  system2 :: Algorithm_descriptor ( )
8  system2 :: init_module_algorithm_parse ( )
9  Initializer :: TypeData :: RunIfNecessary ( Initializer* )
10 Initializer :: RunInitializers ( char const* )
11 RealInit ( char const* , int* , char*** , bool , bool )
12 main
```

The model reduces fragmentation with huge pages by up to 78%.

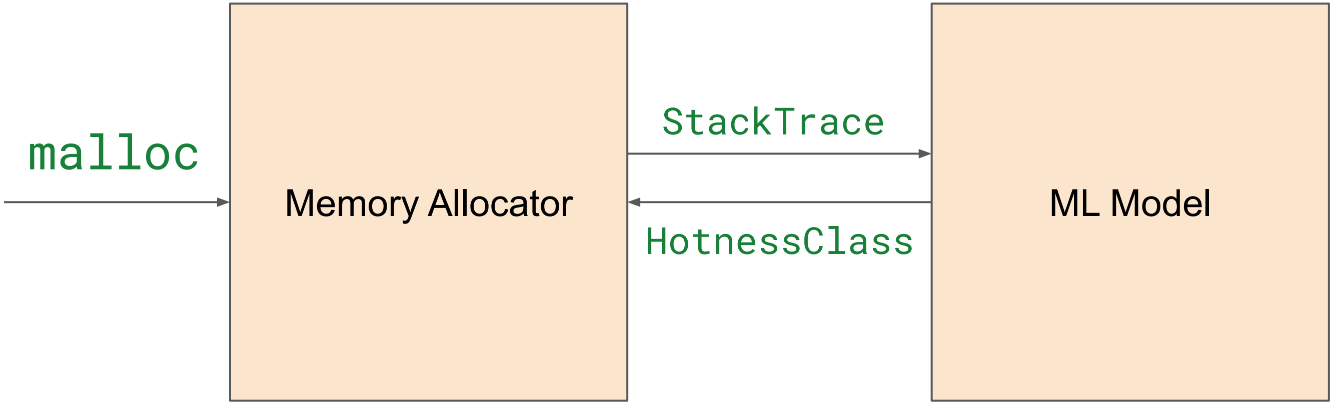
Approach

1. Create an ML-Based **Hotness Prediction**
2. Integrate it with an **Hotness-Aware Allocator**



Hotness Prediction

Hotness Prediction



malloc

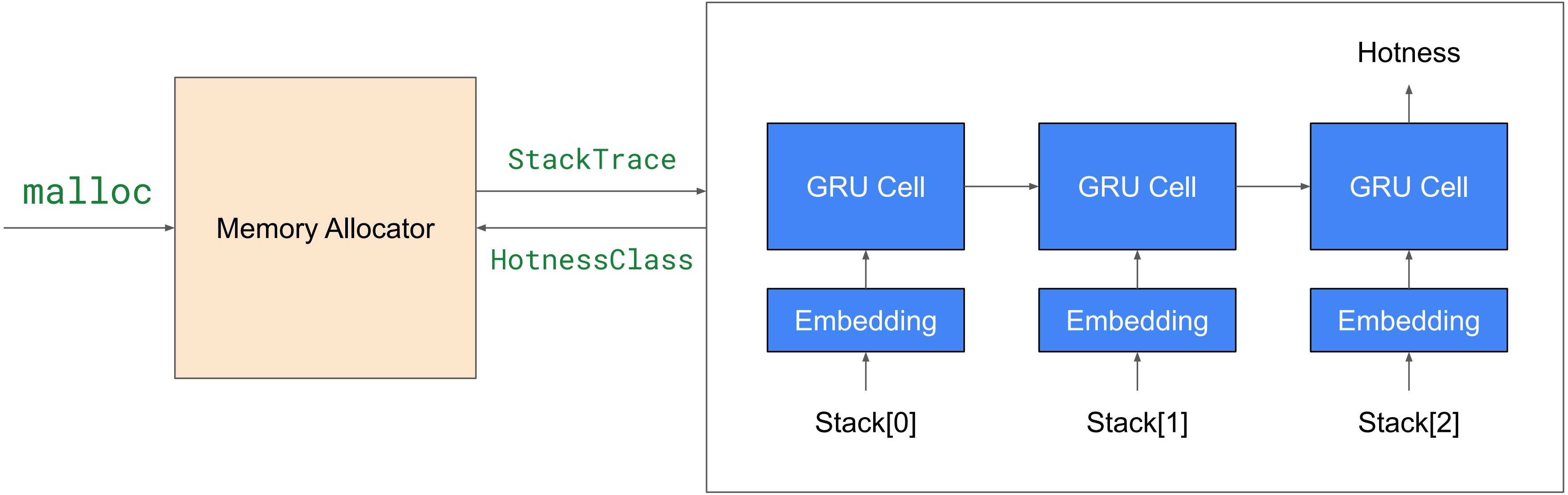
Memory Allocator

StackTrace

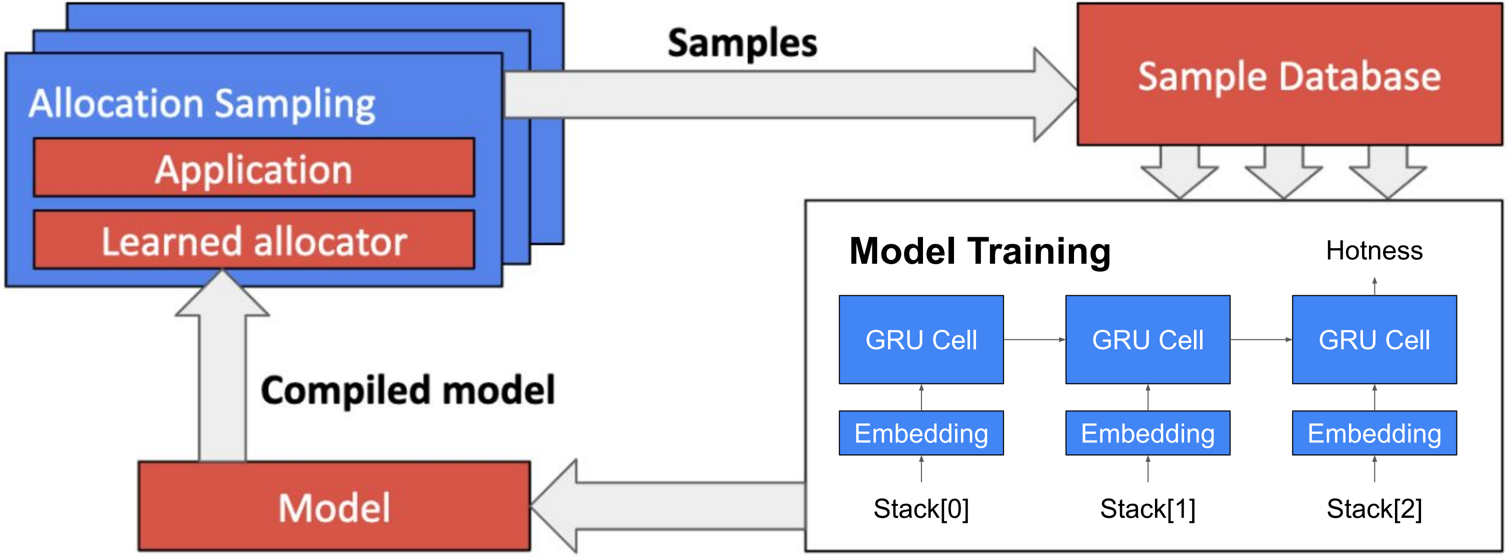
HotnessClass

ML Model

Hotness Prediction

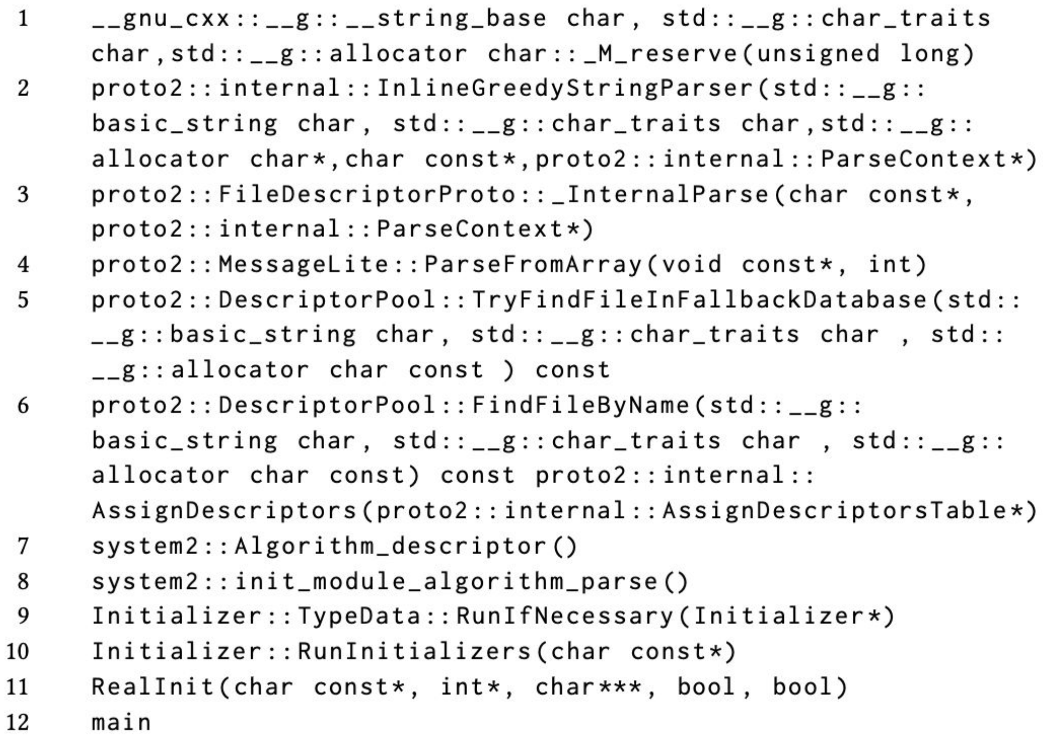


Hotness Prediction



Key Insight

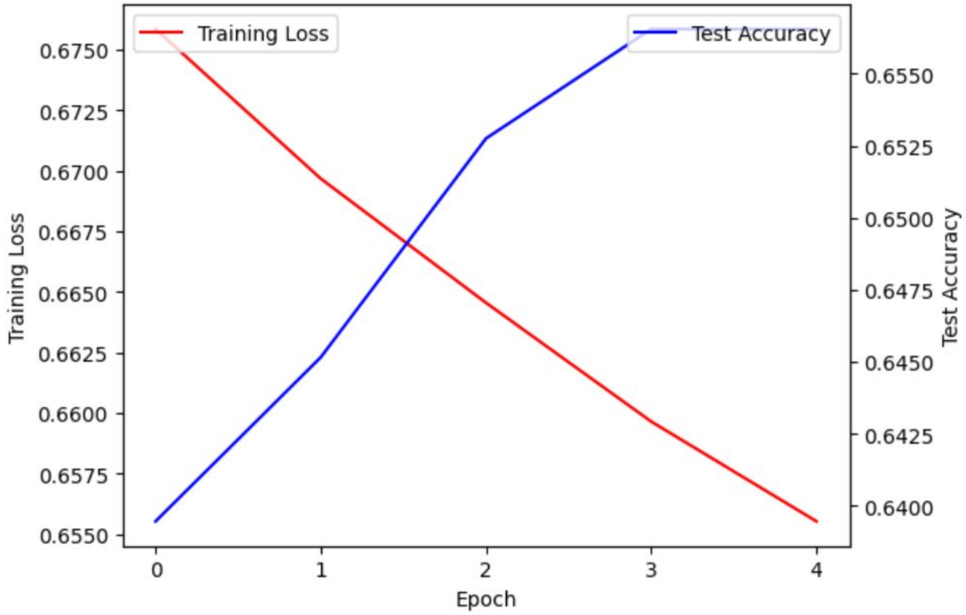
At allocation time, we have access to the stack trace. The program symbols in the stack trace contains *semantic information* about the allocation that is amenable to analysis by a language model.



```
1    __gnu_cxx::__g::__string_base char, std::__g::char_traits
     char,std::__g::allocator char::_M_reserve(unsigned long)
2    proto2::internal::InlineGreedyStringParser(std::__g::
     basic_string char, std::__g::char_traits char,std::__g::
     allocator char*,char const*,proto2::internal::ParseContext*)
3    proto2::FileDescriptorProto::_InternalParse(char const*,
     proto2::internal::ParseContext*)
4    proto2::MessageLite::ParseFromArray(void const*, int)
5    proto2::DescriptorPool::TryFindFileInFallbackDatabase(std::
     __g::basic_string char, std::__g::char_traits char , std::
     __g::allocator char const ) const
6    proto2::DescriptorPool::FindFileByName(std::__g::
     basic_string char, std::__g::char_traits char , std::__g::
     allocator char const) const proto2::internal::
     AssignDescriptors(proto2::internal::AssignDescriptorsTable*)
7    system2::Algorithm_descriptor()
8    system2::init_module_algorithm_parse()
9    Initializer::TypeData::RunIfNecessary(Initializer*)
10   Initializer::RunInitializers(char const*)
11   RealInit(char const*, int*, char***, bool, bool)
12   main
```

Preliminary Results with Incomplete Data

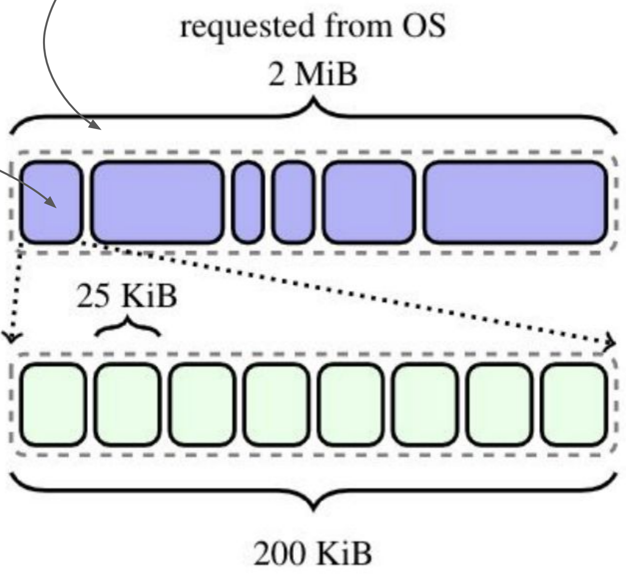
- Trained on small sample
- Shows some predictive power, but more precise accuracy requires larger scale data gathering



Hotness Aware-Allocator

Hotness-Aware Allocator

Store stack trace when spans are allocated



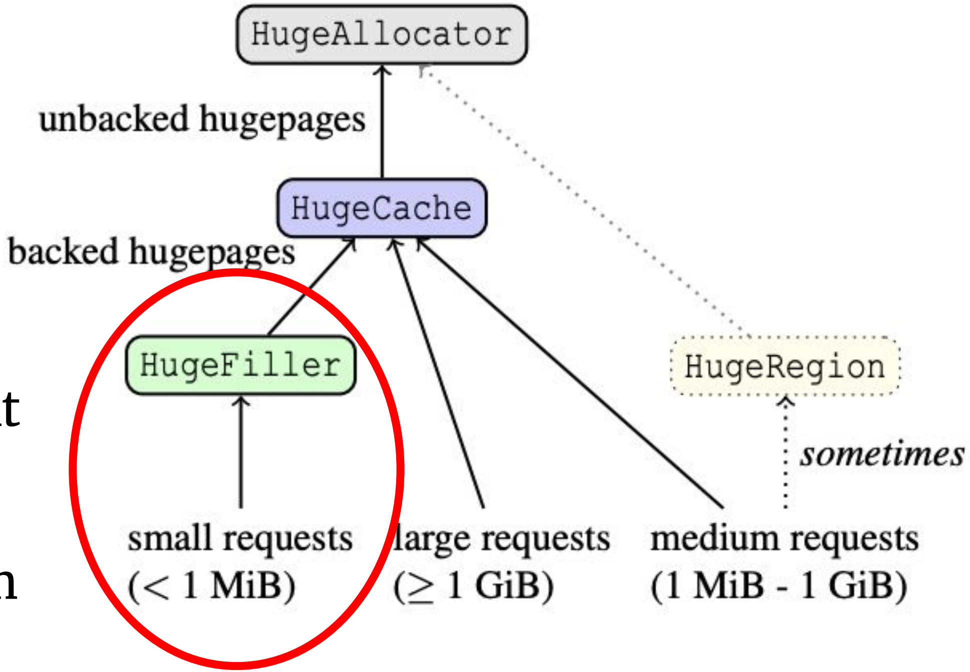
Hugepage

Spans

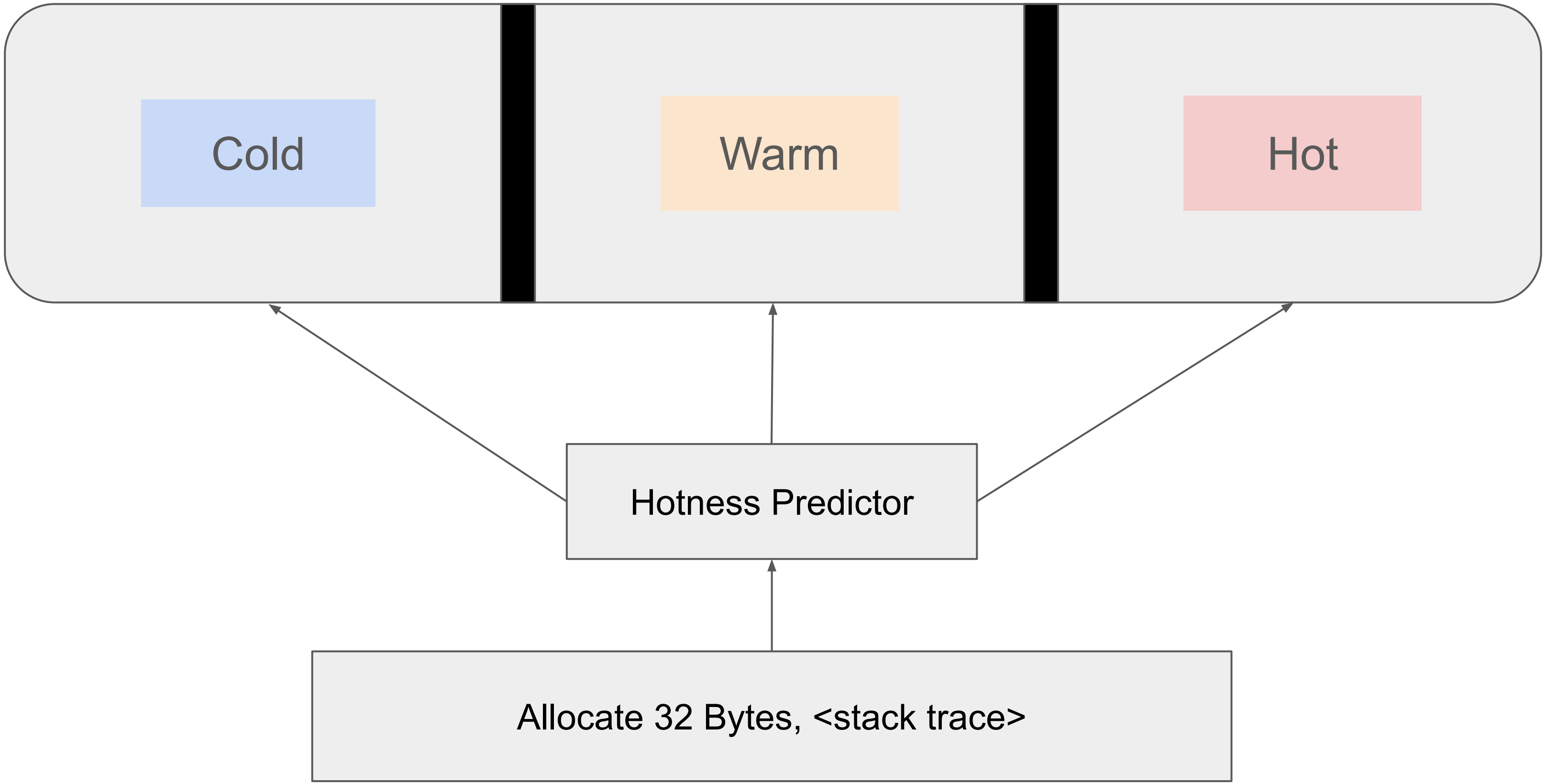
Hotness-Aware Allocator

Divide allocation lists into different hotness levels

Modify algorithm



HugeFiller Partitioning



Concluding Thoughts

Extensions

1. Introduce per-site inference cache
2. Management of prediction errors at runtime