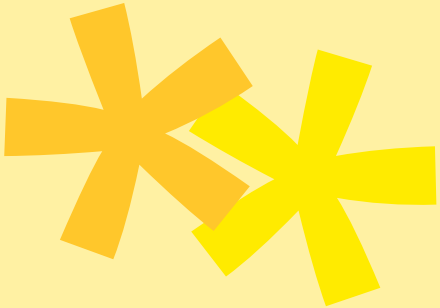


Blink Memory Reduction (@BlinkOn5)

{haraken, bashi, tasak, hajimehoshi}@chromium.org

2015 Nov

Memory team projects

- Oilpan
  - haraken@, keishi@, peria@, yutak@

- Memory reduction (⇐ This talk is about this)
  - haraken@, bashi@, hajimehoshi@, tasak@
  - Working with primiano@, ssid@, ruuda@

- Tab resource reclaiming
  - kouhei@, tzik@

History

- As of 2015 June, memory reduction was raised as one of the priorities of the web-platform team

- At that point, no one knew even what percentage of renderer's memory is consumed by Blink

- Since then, we've been working on understanding Blink's memory and identifying sweet spots to reduce it

Strategy

- Three steps for memory reduction:
  1. Tooling
  2. Understanding
  3. Reducing

Strategy

- Step 1: Tooling
  - Build up memory-infra and provide consistent data to break down the renderer's memory into pieces
  - Provide reproducible telemetry benchmarks and metrics with which developers can keep track of their reduction efforts

Strategy

- Step 2: Understanding
  - Using the tools, identify sweet spots that need reduction

Strategy

- Step 3: Reducing
  - Brainstorm ideas with teams who own the code area
  - Enable the teams to get involved in the reduction
    - Getting more teams involved is key to have a bigger impact

Agenda of this talk

1. Tooling
   - Provide consistent data
   - Provide reproducible benchmarks and metrics
2. Understanding
   - Identify sweet spots
3. Reducing
   - Brainstorm ideas
   - Enable more teams to get involved

Tooling

Allocator unification

- As of 2015 June, Blink mixed four memory allocators
  - PartitionAlloc
  - Oilpan
  - tcmalloc
  - system allocator

- It was crazy...

Allocator unification

- We removed tcmalloc and system allocator from Blink
  - PartitionAlloc
  - Oilpan
  - ~~tcmalloc~~
  - ~~system allocator~~

- Note: Why only from Blink?
  - We experimented with unifying all allocators in Chromium but couldn't observe any clear performance/memory win

memory-infra

- We supported PartitionAlloc and Oilpan in memory-infra
  - We're adding more detailed profiling information to get more understanding

Process	Total resident	Peak total resident	PSS	Private dirty	Swapped	blink_gc	cc	discardable	gpu	gpumemorybuffer	malloc	partition_alloc	skia	v8
Browser (pid 16542)	99.8 MiB	105.5 MiB ↦	68.2 MiB	40.9 MiB	0.0 B		36.4 MiB	0.0 B	10.0 MiB	0.0 B	23.7 MiB		71.7 KiB	2.1 MiB
Renderer (pid 16594)						4.6 MiB		2.6 MiB			4.2 MiB	3.3 MiB	5.5 MiB	13.5 MiB
GPU Process (pid 16605)									6.8 MiB		53.5 MiB			
Renderer (pid 16707): chrome://tracing						3.4 MiB	15.1 MiB	4.0 MiB	3.1 MiB		3.6 MiB	4.7 MiB	132.6 KiB	16.2 MiB
Renderer (pid 16734): Kentaro Hara (@xharaken) | Twitter						9.5 MiB	8.3 MiB	4.7 MiB	3.1 MiB		3.4 MiB	14.3 MiB	18.2 MiB	27.9 MiB
Total	**99.8 MiB**	**105.5 MiB**	**68.2 MiB**	**40.9 MiB**	**0.0 B**	**17.5 MiB**	**59.9 MiB**	**11.3 MiB**	**22.9 MiB**	**0.0 B**	**88.4 MiB**	**22.3 MiB**	**23.9 MiB**	**59.8 MiB**

Allocator details

Allocator	size	effective_size	allocated_objects_size	decommittable_size	discardable_size	slot_size	total_pages_size	virtual_committed_size	virtual_size	active_pages	decommitted_pages	empty_pages
▾partition_alloc	14.3 MiB	14.3 MiB	12.4 MiB	660.0 KiB	252.0 KiB	8.6 MiB	14.4 MiB	17.9 MiB	30.6 MiB	174	114	10
allocated_objects	12.4 MiB	12.4 MiB →										
▾partitions	14.3 MiB ⓘ	2.0 MiB ←	12.4 MiB	660.0 KiB	252.0 KiB	8.6 MiB	14.4 MiB	17.9 MiB	30.6 MiB	174	114	10
▸buffer	7.5 MiB	1.0 MiB	6.0 MiB	660.0 KiB	240.0 KiB	8.5 MiB	11.5 MiB	9.3 MiB	18.6 MiB	87	74	10
▸fast_malloc	4.9 MiB	678.3 KiB	4.5 MiB	0.0 B	12.0 KiB	87.6 KiB	2.1 MiB	6.3 MiB	8.0 MiB	71	38	0
▸layout	2.0 MiB	280.7 KiB	1.9 MiB	0.0 B	0.0 B	3.0 KiB	828.0 KiB	2.3 MiB	4.0 MiB	16	2	0
node	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B			0.0 B	0.0 B			

More detailed profiling

- What types of objects and how many objects are allocated in Blink (per-object-type profiler)

Object type	Size	Count
Vector<RuleData>	12256 bytes	3
Vector<RuleFeature>	800 bytes	6
ImmutableStylePropertySet	26 bytes	2112
...	...	...

More detailed profiling

- Where large objects are allocated (allocation-site profiler)

```
StringImpl 1020 KB
    <--- AtomicString::AtomicString()
    <--- ScriptResource::script()
```

More detailed profiling

- Retaining relationships that cross allocator boundaries
  - Blink retains a lot of memory in other allocators in the renderer process

`HTMLLinkElement::m_resource => locked discardable memory (620 KB)`

`FontCustomPlatformData::m_typeface => Skia (700 KB)`

Q4/Q1 Roadmap

- Add more profiling information to memory-infra
  - Per-object-type profiler
  - Allocation-site profiler
  - Cross-allocator retaining relationships

- Add reproducible telemetry benchmarks and metrics with which developers can keep track of their reduction efforts

Understanding

Scope

- Scope:
  - Understand the breakdown of renderer's memory

- The following slides are focusing on memory only inside Blink, but our scope is to understand and reduce renderer's memory (especially renderer's memory retained by Blink)

Scope

- Short-lived apps or long-lived apps?
  - For short-lived apps: memory after 10 seconds after a page load matters
  - For long-lived apps: memory after 1 hour after a page load & a lot of user interactions matters

- We’re now focusing on short-lived apps because short-lived apps would be more important in mobile devices, but long-lived apps are on our radar too

Break down the memory into pieces

- Break down renderer's memory
  - Renderer = V8 + PartitionAlloc + Oilpan + Skia + …
- Break down the PartitionAlloc's memory
  - PartitionAlloc = Buffer + FastMalloc + Node + Layout
- Break down each partition into per-object memory
  - Buffer = StringImpl + Vector + HashTable + …
  - FastMalloc = SharedBuffer + ...

Break down of renderer's memory

Page	Total (*1)	PartitionAlloc	Oilpan	V8	Skia	Discardable	cc	malloc
pinterest.com	78.06 MB	34.35%	1.76%	13.32%	4.32%	20.50%	17.93%	7.82%
facebook.com	70.13 MB	33.02%	1.25%	8.63%	16.64%	17.11%	14.97%	8.38%
worldjournal.com	56.71 MB	41.57%	1.76%	11.24%	6.19%	7.05%	18.74%	13.45%
wikipedia.org	44.32 MB	34.82%	2.26%	11.92%	2.28%	9.03%	23.69%	16.01%
reddit.com	72.10 MB	21.15%	1.56%	9.72%	14.08%	27.74%	14.63%	11.12%
wordpress.com	77.31 MB	26.83%	1.78%	31.67%	5.10%	5.17%	14.55%	14.89%
plus.google.com	41.84 MB	27.11%	2.99%	13.98%	3.66%	19.12%	14.11%	19.03%
blogspot.com	57.57 MB	25.23%	1.74%	32.35%	3.43%	6.95%	18.24%	12.08%
mail.google.com	50.73 MB	17.91%	1.72%	7.10%	16.39%	23.65%	21.07%	12.15%

Break down of renderer's memory

- In common cases V8 is the largest consumer, but in these pages Blink is the largest consumer (20 - 40%)
  - because the page set is chosen that way (i.e., they are key 10 pages where Blink's reduction is a key)

- Some (a lot?) of the memory in Discardable, CC and Skia is retained by Blink

Break down of Blink's memory

- Since we've not yet fully shipped Oilpan, most Blink objects are allocated in PartitionAlloc

- PartitionAlloc has four partitions:
  - Node (for Nodes)
  - Layout (for LayoutObjects)
  - Buffer (for Vectors, HashTables, StringImpls, ArrayBuffers etc)
  - FastMalloc (for SharedBuffers and all other objects)

Break down of PartitionAlloc's memory

Page	Total of PartitionAlloc	Buffer	FastMalloc	Node	Layout
pinterest.com	26.76 MB	75.74%	21.93%	1.17%	1.17%
facebook.com	23.30 MB	35.90%	58.46%	1.44%	4.19%
worldjournal.com	23.16 MB	34.08%	40.92%	9.11%	15.89%
wikipedia.org	15.56 MB	37.25%	37.15%	9.74%	15.86%
reddit.com	15.26 MB	47.88%	39.89%	3.89%	8.35%
wordpress.com	20.92 MB	52.88%	38.61%	4.33%	4.18%
plus.google.com	11.41 MB	66.19%	27.10%	3.08%	3.63%
blogspot.com	13.86 MB	51.69%	36.87%	3.55%	7.89%
mail.google.com	9.14 MB	58.46%	31.71%	3.33%	6.50%

Break down of PartitionAlloc's memory

- The largest consumer is the Buffer partition
  - StringImpls, Vectors, HashTables, ArrayBuffers etc

- The second largest consumer is the FastMalloc partition
  - SharedBuffers, all DOM objects except for Nodes and LayoutObjects

- sizeof(Node) and sizeof(LayoutObject) don't really matter

Break down of the Buffer partition

Page	Total of the Buffer partition(*1)	StringImpls	Vectors	HashTables	Others
pinterest.com	9.51 MB	90.19%	2.64%	6.98%	0.19%
facebook.com	2.91 MB	41.23%	40.50%	17.00%	1.26%
worldjournal.com	3.22 MB	45.61%	39.00%	14.68%	0.71%
wikipedia.org	2.00 MB	65.18%	7.30%	26.63%	0.89%
reddit.com	2.89 MB	41.21%	42.39%	16.21%	0.20%
wordpress.com	4.94 MB	55.32%	29.88%	14.50%	0.29%
plus.google.com	2.62 MB	81.56%	7.23%	10.77%	0.44%
blogspot.com	2.89 MB	66.53%	14.02%	17.88%	1.57%
mail.google.com	2.11 MB	30.77%	54.24%	13.99%	1.00%

Break down of the Buffer partition

- The largest consumer is StringImpls
  - We confirmed that more than 90% of the large StringImpls are JavaScript source code

- The second largest consumer is Vectors and HashTables
  - We confirmed that 25% of the Vectors are consumed by the unused region
  - We confirmed that 70% of the HashTables are consumed by the unused region of small HashTables

Break down of the FastMalloc partition

Page	Total of the FastMalloc partition	SharedBuffers	Other objects
pinterest.com	2.38 MB	15.93%	84.07%
facebook.com	6.45 MB	74.69%	25.31%
worldjournal.com	2.90 MB	53.12%	46.88%
wikipedia.org	2.19 MB	6.25%	93.75%
reddit.com	2.32 MB	13.47%	86.53%
wordpress.com	3.58 MB	27.36%	72.64%
plus.google.com	0.92 MB	13.22%	86.78%
blogspot.com	2.32 MB	22.88%	77.12%
mail.google.com	1.01 MB	0.00%	100.00%

Break down of the FastMalloc partition

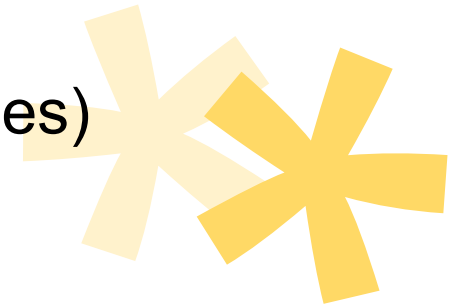
- The largest consumer is SharedBuffers
  - They are used as backing storage of Resources

- Other objects have a very long tail
  - The 2nd - 5th largest consumers depend on webpages
  - Reducing sizeof(DOM object) won’t contribute to reducing Blink’s memory

Summary

- The largest consumers inside Blink are:
  1. StringImpls
  2. SharedBuffers
  3. Vectors + HashTables

- BTW, how much memory is consumed by Vectors and HashTables as absolute values?
  - Only < 1 MB... (even in the Blink-heavy pages)
  - It’s not worth reducing...



What does it mean?

- Inside Blink, the only objects we should really reduce would be StringImpls and SharedBuffers

- Reducing other objects won’t contribute to reducing renderer’s memory

What does it mean?

- After reducing StringImpls and SharedBuffers, we should look at outside Blink, not inside Blink
  - “Outside Blink” consumes 60 - 80% (even in the Blink-heavy pages)
  - “Outside Blink” doesn’t mean that Blink is not related
    - Blink retains a bunch of memory in Discardable, Skia, CC etc

- Remember that our goal is to reduce renderer’s memory

What we have learned

- Memory reduction based on a “guess” is NOT likely to contribute to a meaningful reduction
  - Reducing sizeof(DOM object) won’t have an impact
  - We experimented with discarding various Blink caches each TL listed in this spreadsheet, but most of them didn’t have an impact

What we have learned

- It is important to drive the reduction efforts more tactically
  1. Understand the breakdown
  2. Identify key problem areas (sweet spots)
  3. Reduce the memory at a pinpoint

What we have learned

- It is really hard to collect consistent and reproducible results
  - It is a substantial amount of work to confirm the reasonableness of various memory metrics
  - Example:
    - Render’s private memory >> sum(memory usage of each allocator)
      - Where does the dark matter come from…?
    - Various “size” notions
      - resident, effective, active, committed...

Q4/Q1 roadmap

- Get more understanding on outside Blink
- Understand the real world
  - UMAs
  - Deep reports
  - OOM
- Identify X projects that would have the biggest impact
- Profile long-running apps

Reducing

Strategy

- Our strategy is:
  - Provide good tooling and understanding
  - Brainstorm ideas for reduction
  - And enable more teams to get involved in the reduction (instead of only the memory team working on the reduction)
    - That way we can move faster

On-going reduction projects

- MemoryPurgeController (⇒ Memory team)
- CompressableString (⇒ Memory team)
- More aggressive Resource pruning (⇒ Loading team)
- Making V8 ⇔ Blink cycles collectable (⇒ V8 team)

MemoryPurgeController

- When Blink should save memory, MemoryPurgeController dispatches purgeMemory()

```
class MemoryPurgeController {
  void purgeMemory(...) { // Dispatched when Blink should save memory
    for (auto& client : m_clients)
      client->purgeMemory(...);
  }
  HashSet<MemoryPurgeClient*> m_clients;
};
class MemoryCache : public MemoryPurgeClient {
  void purgeMemory(...) override { pruneAll(); } // Example
};
```

MemoryPurgeController

- When does MemoryPurgeController dispatch the purgeMemory()?
  - After 10 seconds after a tab goes inactive
  - When Blink receives MemoryPressureListener's notifications

MemoryPurgeController

- What should the purgeMemory() do?
  - Discard discardable caches in Blink

- Even though the purgeMemory() is dispatched against only inactive tabs & memory-pressure situations, we must be careful about its performance impact

CompressableString

- The largest consumer of PartitionAlloc is large Strings

- More than 90% of the large Strings are JavaScript source code
  - They are not discardable (used by V8)
  - The only option to reduce the memory is to compress the Strings (⇒ Introduce CompressableString)

CompressableString

- We confirmed that if we gzip large Strings, we can reduce PartitionAlloc's memory by 10 - 60% (17% in average)

Page	Reduction rate
pinterest.com	63%
facebook.com	15%
theverge.com	11%
worldjournal.com	3%
wikipedia.org	19%
reddit.com	10%
wordpress.com	14%
plus.google.com	5%
blogspot.com	9%
mail.google.com	21%

More aggressive Resource pruning

- The second largest consumer of PartitionAlloc is SharedBuffers

- SharedBuffers are used as backing storage of Resources
  - Memory reduction of Resources is important

More aggressive Resource pruning

- Actually, memory reduction of Resources is more important than the data implies, because Resources also retain memory in Discardable and Skia
  - Some Resources use a locked memory in Discardable
  - Image/FontResources can hold caches in Skia

More aggressive Resource pruning

- Low-hanging fruits:
  - HTMLLinkElement should clear CSSStyleSheetResource when it finishes parsing the stylesheet
    - This reduces 1.2 MB from pinterest.com, 800 KB from theverge.com
  - GlyphPage should discard GlyphPageTree
    - This reduces 200 KB from theverge.com
  - FontCustomPlatformData should discard cached font-faces in Skia

More aggressive Resource pruning

- Goals:
  - Don’t keep alive ResourcePtrs longer than needed
  - Discard various caches retained by Resources (unless it regresses performance)
  - Visualize the retaining relationships in memory-infra

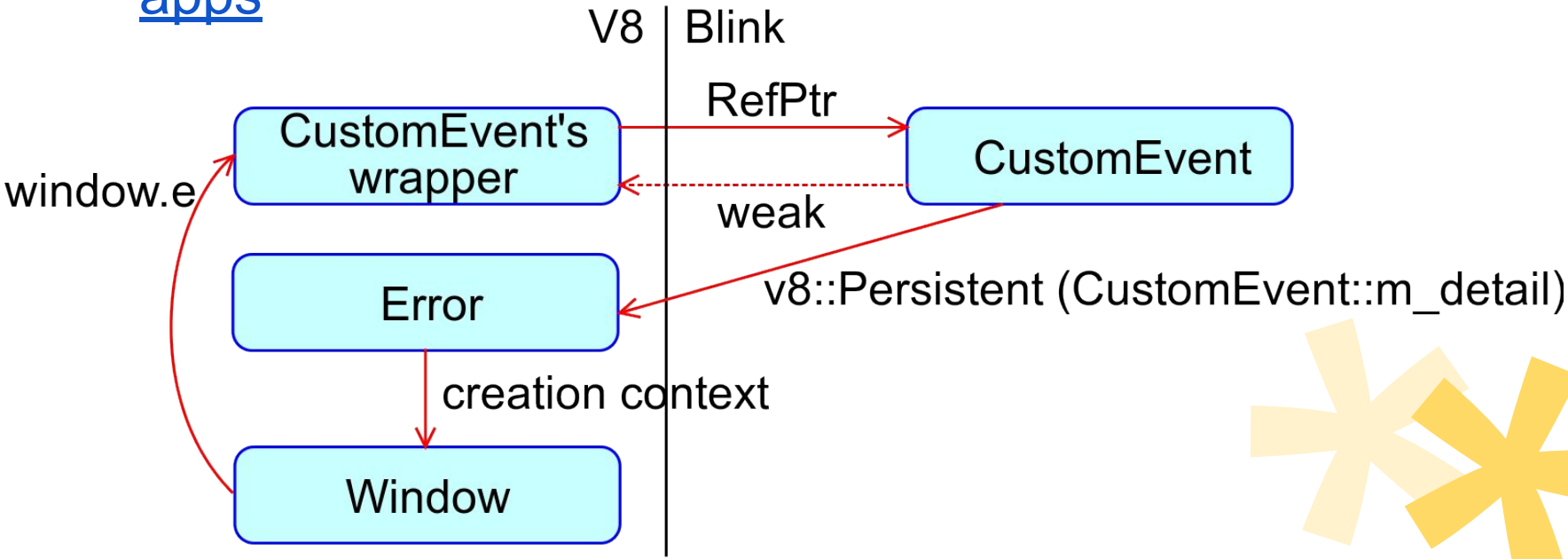
Making V8 ⇔ Blink cycles collectable

- In common cases, V8 is the largest consumer of renderer's memory

- At the very least, Blink should not be the culprit of V8 memory leaks

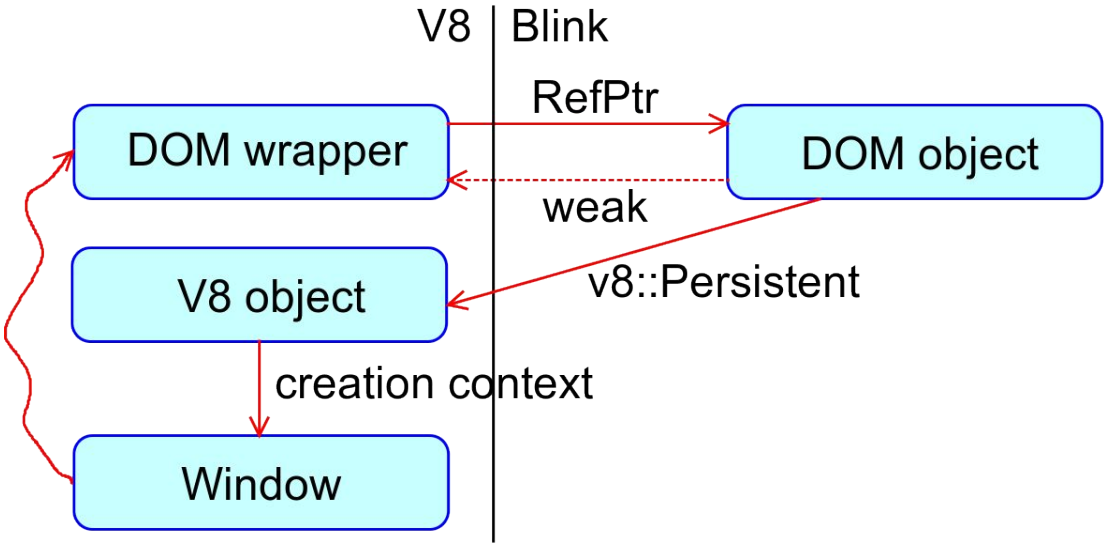
Making V8 ⇔ Blink cycles collectable

- Example: `e = new CustomEvent("foo", {detail: new Error()});`
  - This was a cause of [a lot of window leaks in Polymer apps]



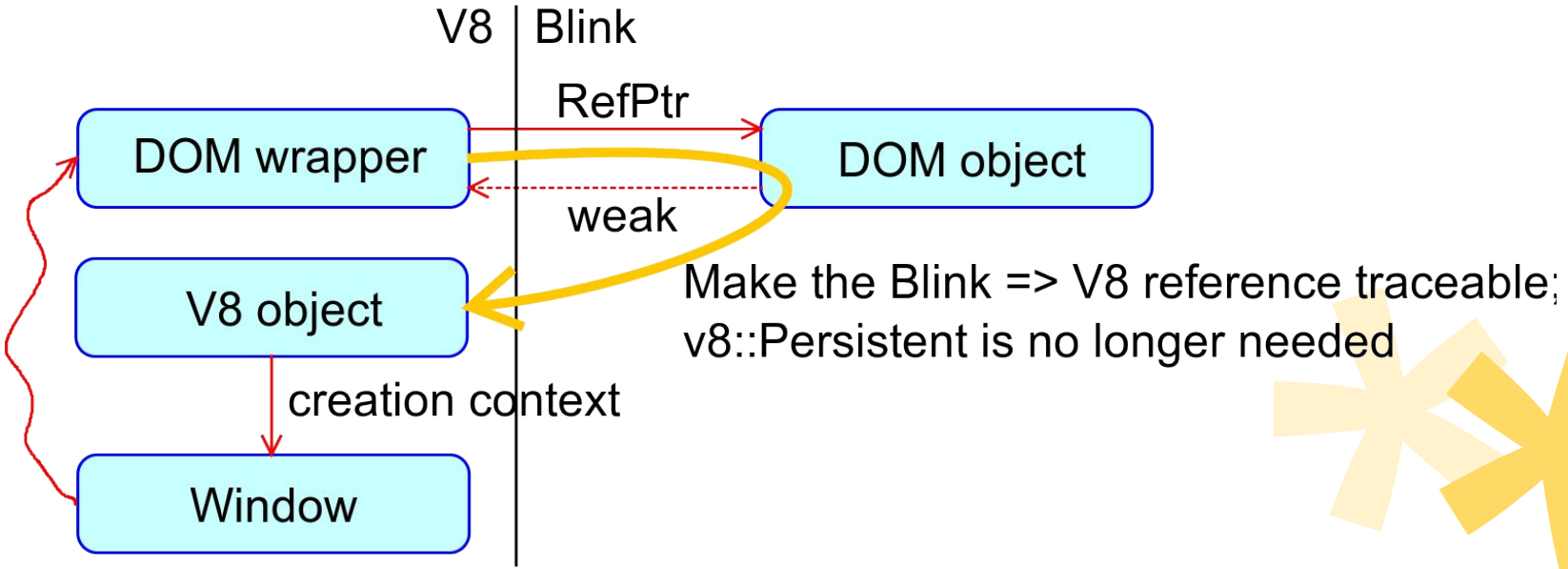
Making V8 ⇔ Blink cycles collectable

- In general, a V8 object has a strong reference to the window object that created the V8 object
- Thus if a Blink object retains a v8::Persistent handle to the V8 object, it retains the entire window object...



Making V8 ⇔ Blink cycles collectable

- Idea: Remove all v8::Persistent handles by making V8 ⇔ Blink cycles collectable
  - All V8 leaks caused by Blink will be gone



Summary

- Various reduction projects are going on:
  - MemoryPurgeController
  - CompressableString (for StringImpls)
  - More aggressive Resource pruning (for SharedBuffers)
  - Making V8 ⇔ Blink cycles collectable

- As we identify more key problem areas, we'll work with teams who own the area and brainstorm ideas for reduction

Q4/Q1 roadmap

- Establish the following development flow:
  - Provide good tooling
  - Identify key problem areas
  - Work with teams who own the area and brainstorm reduction ideas
  - Enable the teams to set memory-reduction OKRs and work on the reduction

Conclusions

Scope

- Understand and reduce renderer’s memory

Current understanding

- In Blink-heavy pages, Blink can be the largest consumer (20 - 40%)
  1. StringImpls
  2. SharedBuffers
  3. Vectors and HashTables

- After reducing StringImpls and SharedBuffers, we should look at outside Blink (some of them are retained by Blink)

Strategy

1. Tooling
   - Provide consistent data in memory-infra
   - Provide reproducible telemetry benchmarks and metrics
2. Understanding
   - Identify sweet spots
3. Reducing
   - Brainstorm ideas
   - Enable more teams to get involved

Strategy

- Our goal is to establish a development flow that enables each team to set memory-reduction OKRs and get involved in the reduction
  - Instead of only the memory team working on the reduction :)

- Empowering TRIM for that direction!