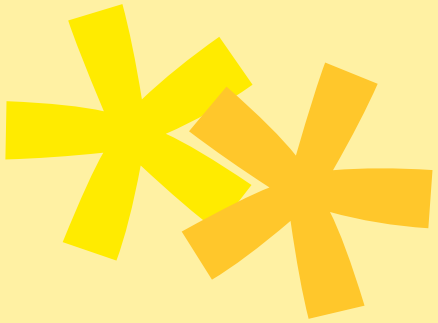


Chromeのなかの コンピュータ・サイエンス

haraken@chromium.org

2015 Sep

今日のお話

- エンジニアリングの仕事というと、「ユーザの目に触れるプロダクト作り」や「新しい機能のローンチ」ばかりが注目されがちですが、それればかりがエンジニアリングではありません

- プロダクトの内側にはコンピュータ・サイエンスの世界が広がっていて、その進化がプロダクトの進化を支えています
  - ふだんはあまり語られることのない（地味な）世界ですが、今日はその魅力について語ります！

今日のお話

- 「Chromeのなかのコンピュータ・サイエンス」ということで、Garbage Collection (GC; ゴミ集め) というしくみについて、非エンジニアの方々向けにわかりやすく解説します

- GCは現在でも最先端で研究されているコンピュータ・サイエンスの基礎技術です

今日のお話

- ひと昔前のChromeを使っていたとき、こんな経験をしたことはないでしょうか？
  - Gmailが数秒間固まった
  - Google Mapsを指でグイーーーーンと移動させたときにカクカクした

- あのカクカクの原因のほとんどがGCでした
  - Chromeをサクサク動かすためには、GCをいかに賢く作り込んでいくかが非常に重要な鍵になります

今日のお話

- 今日のお話で伝えたいこと:
  - いったいGCとはどんなものなのか
  - どうしてそんなものがChromeに必要なのか
  - 何がそんなにおもしろいのか

- プロダクトの内側に広がっているコンピュータ・サイエンスの世界の「おもしろさ」と「意義」を伝えられればと思います

いったいGCとはどんなものなのか

まずは基本的な話から

- コンピュータでプログラムを実行するためには、たくさんのデータが必要になります

- たとえば、ChromeでWebページを開いたとき、Webページを表示するためにはいろいろなデータ（文字、画像、ボタン、音声など）が必要です

まずは基本的な話から

- Chromeで10個のWebページを順番に開いたときのことを考えてみてください
  1. 1個目のWebページを開く ⇒ たくさんのデータが必要
  2. 2個目のWebページに移動する ⇒ たくさんのデータが必要
  3. …

- これを繰り返していくと、Webページを開くたびにデータがどんどんコンピュータ上に蓄積されていって、やがてはメモリを使い尽くしてしまいます（Chromeがクラッシュします）

まずは基本的な話から

- これを防ぐにはどうすればよいでしょうか？
  - （あたりまえですが）すでに見終わったWebページのデータを捨てるようにすればOKです

- そうすれば、不要になったデータがどんどん蓄積していってメモリを使い尽くす事態を防げます

まずは基本的な話から

- （あたりまえですが）重要な事実：
  - 不要になったデータを捨てずにChromeを実行しつづけたら、やがてメモリを使い尽くしてクラッシュしてしまう
  - つまり、不要になったデータは捨てないといけない

不要なデータをどう捨てるか

- 「不要なデータをどう捨てるか」というのは、それほど自明な問題ではありません
  - 「Webページを移動するときに、前のWebページのデータを全部捨てる」というのはわかりやすい方法ですが、これだけでは不十分です
  - というのは、Webページを移動しなくても、Webページ内に「動き」があれば不要なデータは絶えず作られているからです（ページ内動画、音声、すぐに切り替わる広告など）

不要なデータをどう捨てるか

- もし「Webページ移動のときしか不要なデータを捨てない」ことにしてしまうと、「動き」のあるWebページを放置しておくだけで、どんどんデータが蓄積していって、メモリを使い尽くすことになります
  - Gmailを1時間開きっぱなしにすることができなくなる

- つまり、Webページの移動の有無に関係なく、「定期的に不要なデータを捨てる」仕組みが必要になります

不要なデータをどう捨てるか

- ところが、「定期的に不要なデータを捨てよう」とすると難しい問題が出てきます
  - Webページ移動のときなら、前のWebページのデータを全部捨てればいいだけの話でしたが…
  - Webページが動いている途中で不要なデータを捨てようと思ったら、「どのデータが必要でどのデータが不要か」を見分ける手段が必要になります

必要なデータと不要なデータをどう見分けるか？

- 単純な例として、「前の写真」「次の写真」をクリックすると（Webページが移動することなく）写真が切り替わるWebサイトを考えてみましょう

必要なデータと不要なデータをどう見分けるか？

- ひとまず、データの必要・不要を「いま表示されているかどうか」で定義することにしましょう

いま表示されているこの画像は必要

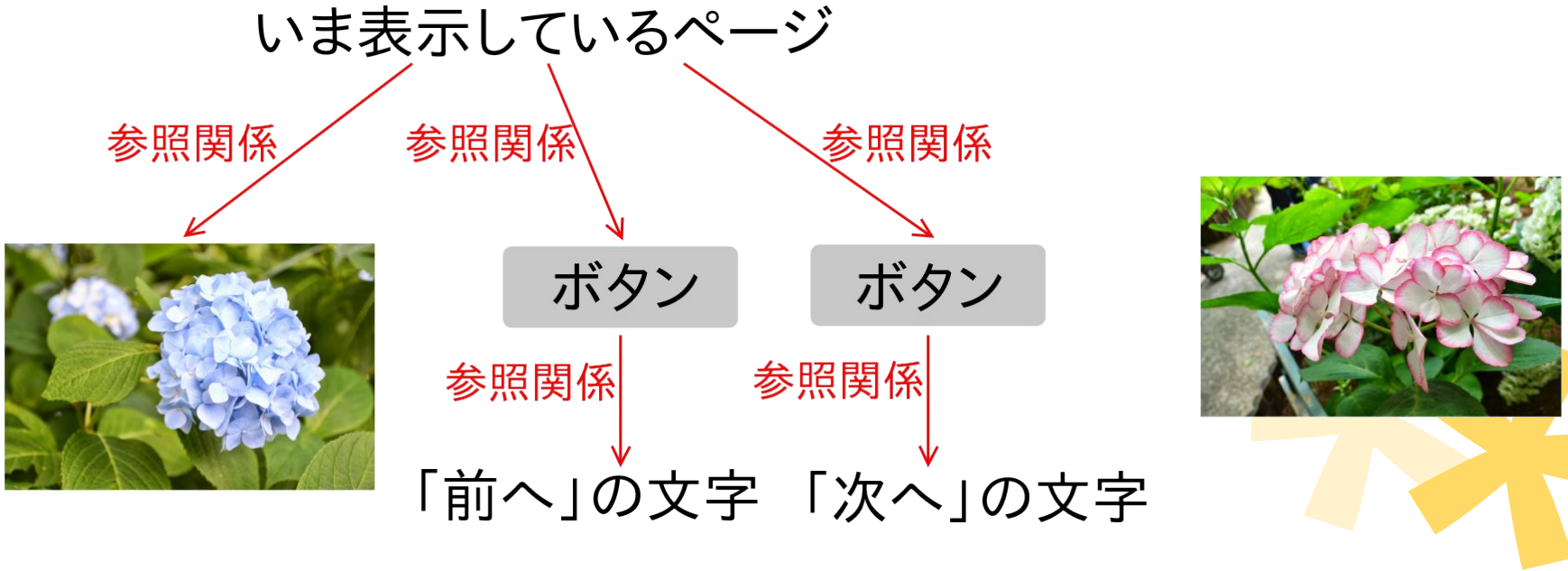
直前に表示されていた
この画像はもう不要

いま表示されている
このボタンは必要

いま表示されている
この文字は必要

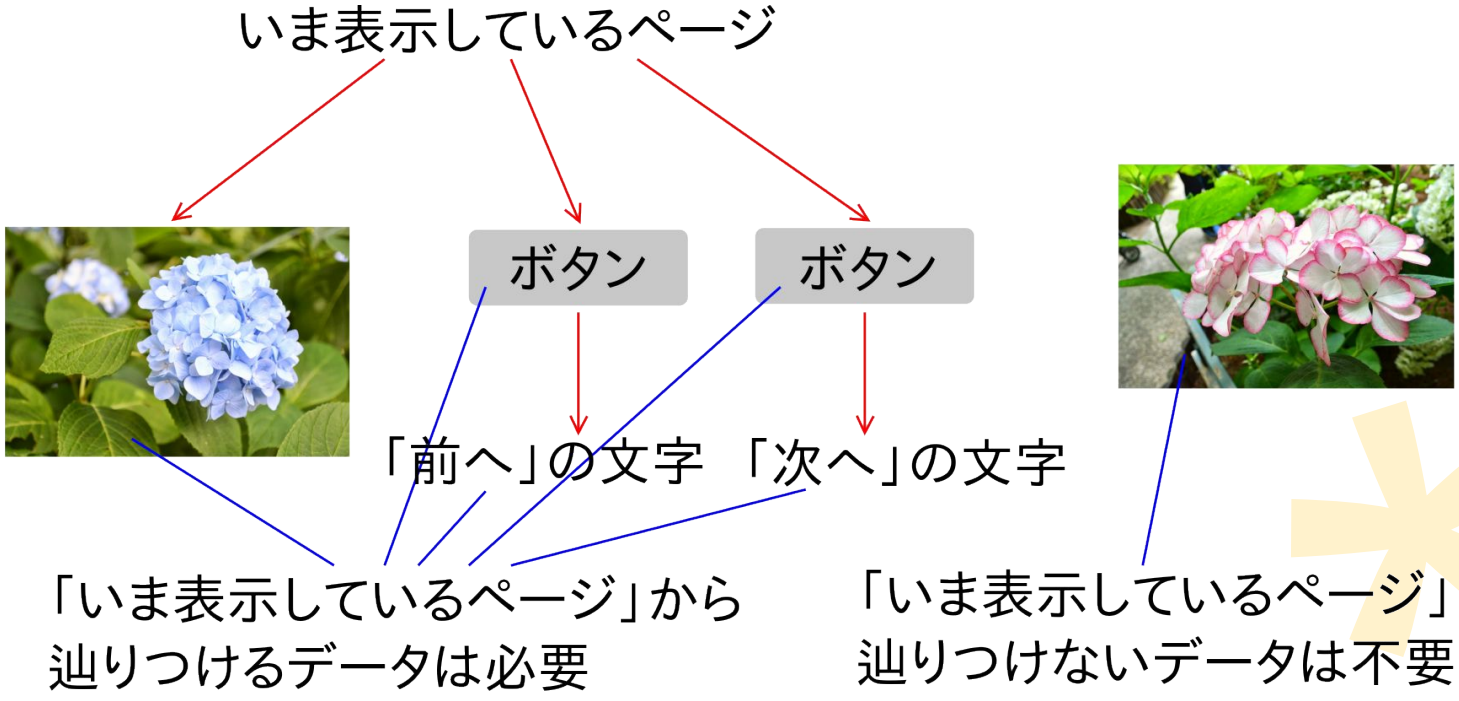
必要なデータと不要なデータをどう見分けるか？

- もう少し厳密に考えるために、データどうしの参照関係をモデル化してみます



必要なデータと不要なデータをどう見分けるか？

- この参照関係を辿れば、必要なデータと不要なデータを見分けられますね



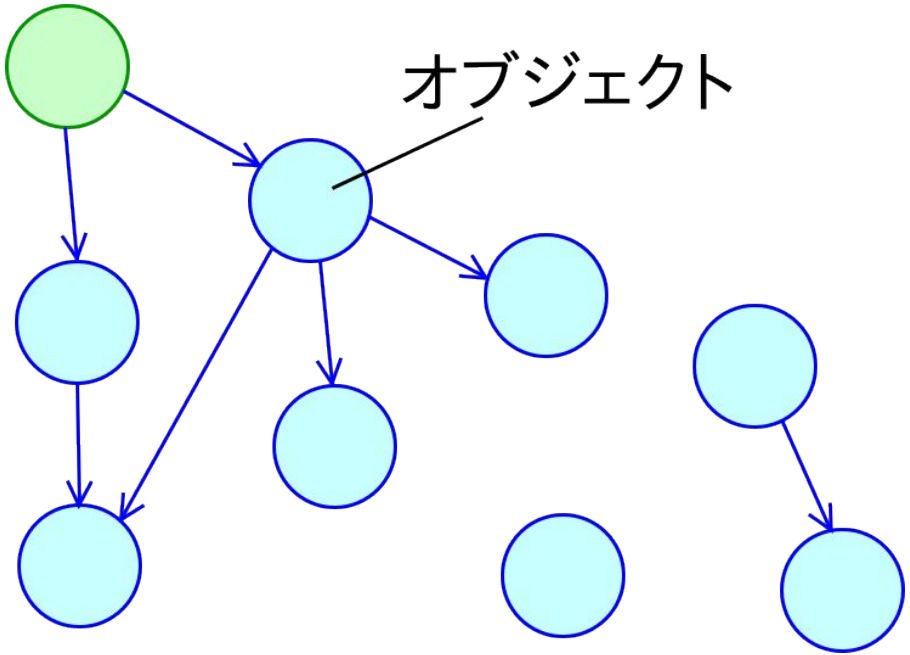
必要なデータと不要なデータをどう見分けるか？

- 要するに、データどうしの参照関係を矢印で表現したうえで、以下のように必要・不要を判断すればよいわけです
  - 「いま表示しているページ」から辿りつけるデータは必要
  - 「いま表示しているページ」から辿りつけないデータは不要

- 以上の話を一般化してみましょう

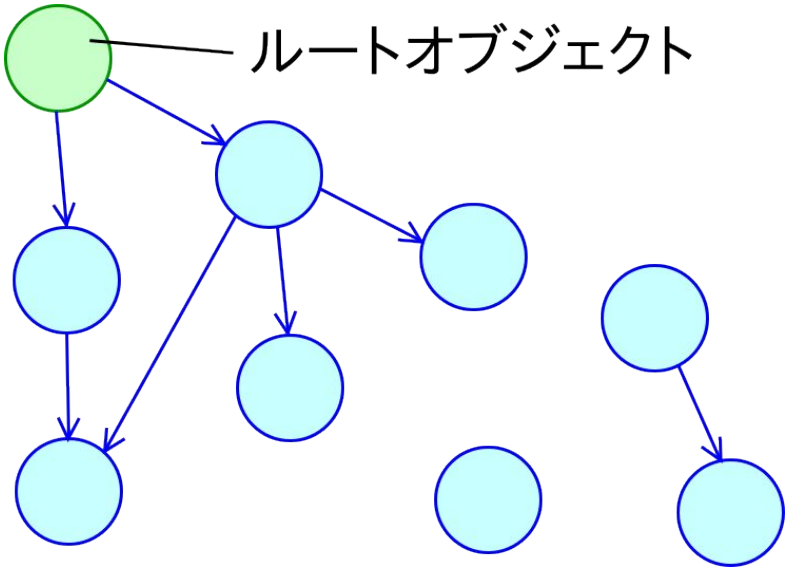
問題の一般化

- GCの世界では、各データのことを「オブジェクト」と呼びます
- オブジェクトどうしの参照関係全体のことを「オブジェクトグラフ」と呼びます



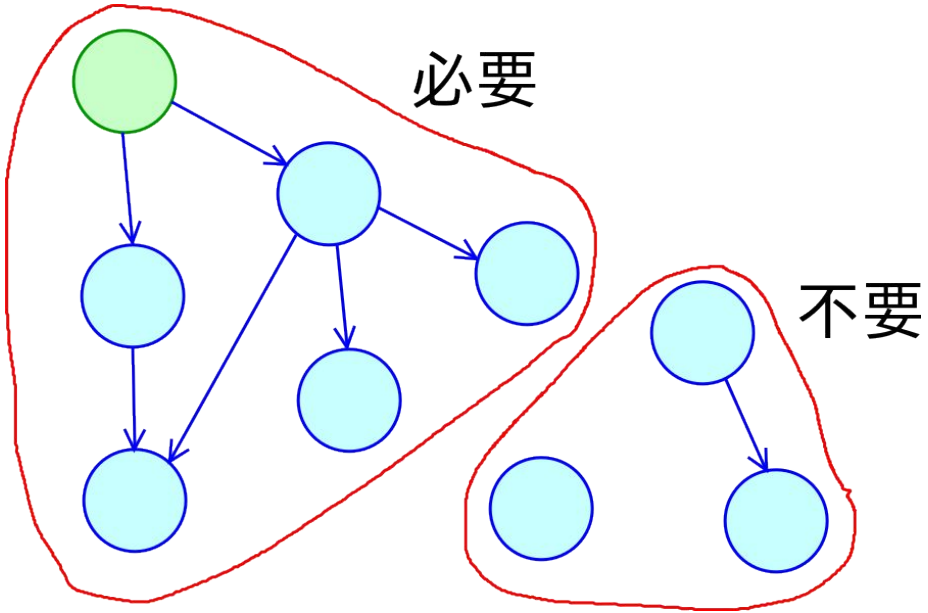
問題の一般化

- グラフをたどる出発点となるオブジェクトのことを「ルートオブジェクト」と呼びます
  - さっきの例では、「いま表示しているWebページ」がルートオブジェクト



問題の一般化

- オブジェクトの必要・不要は次のように判断できます：
  - ルートオブジェクトから辿りつけるものは必要（捨てちゃだめ）
  - ルートオブジェクトから辿りつけないものは不要（捨ててOK）



これがGCなのだ！！

- ・・・という（おそらくChromeユーザの99.9%はまったく意識したこともない）一連の作業をやってくれる仕組みこそがGCなのです

- Garbage Collection (GC; ゴミ集め) ＝ オブジェクトグラフを辿って各オブジェクトの必要・不要を判断し、不要なオブジェクトを捨ててくれる仕組みのこと

ここまでのまとめ

- 不要になったデータを捨てずにChromeを実行しつづけたら、やがてメモリを使い尽くしてしまう
  - そこで「定期的に不要なデータを捨てる」仕組みが必要

- GCは、データの参照関係をオブジェクトグラフとして定義し、グラフを辿って各オブジェクトの必要・不要を判断して、不要なオブジェクトを捨ててくれる

昔のGoogle Mapsはなぜカクカクしたのか

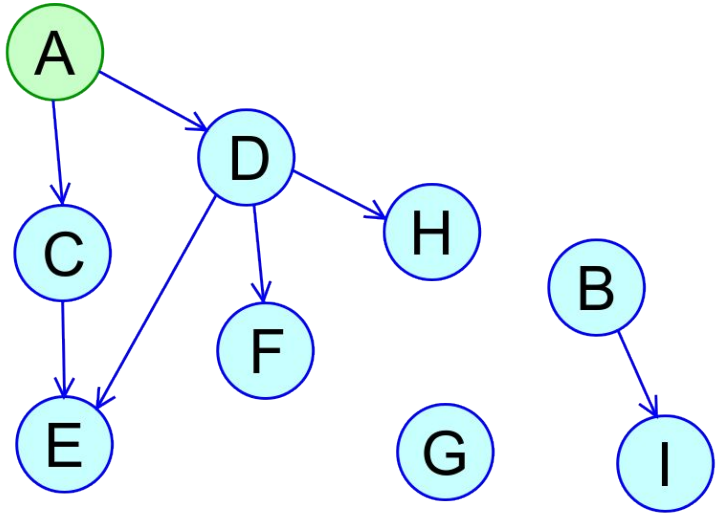
昔のGoogle Mapsはなぜカクカクしたのか

- ここまでの説明で、GCが何をするものかはわかりました

- 次に、もっと具体的にGCがどう動くものなのかを考えます
  - これがわかると、なぜひと昔前のGmailやGoogle Mapsがカクカクしていたのかの理由が見えてきます

具体的なGCの仕組み

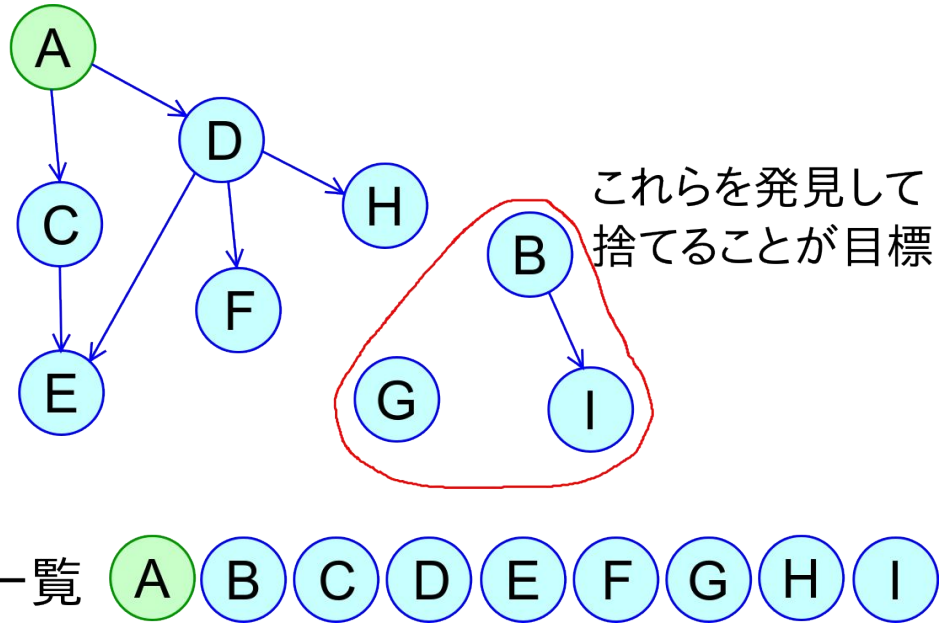
- GCが管理する情報は2つ:
  - 「オブジェクトグラフ」と「オブジェクトの一覧」



一覧 A B C D E F G H I

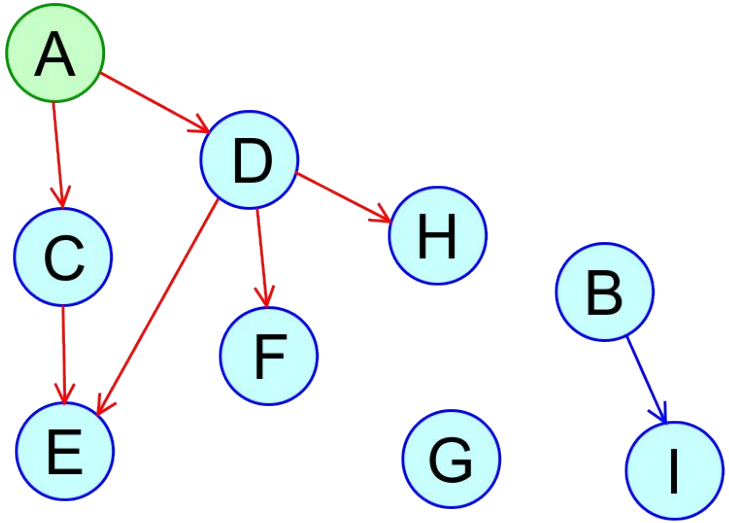
具体的なGCの仕組み

- 目標は、「オブジェクトの一覧のなかから、グラフ上で辿りつけないオブジェクトを発見して捨てること」
  - どうやればよいか？



具体的なGCの仕組み

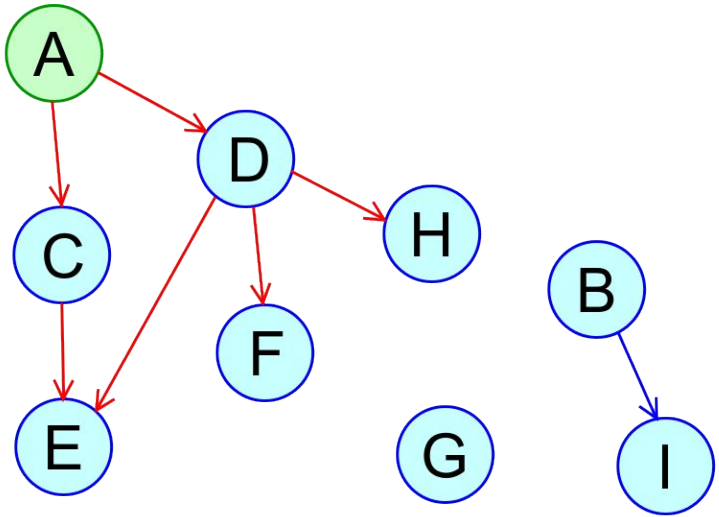
- 手順1：ルートオブジェクトからグラフを辿って、辿ったオブジェクトにマークを付ける（「マーキング」と呼ぶ）

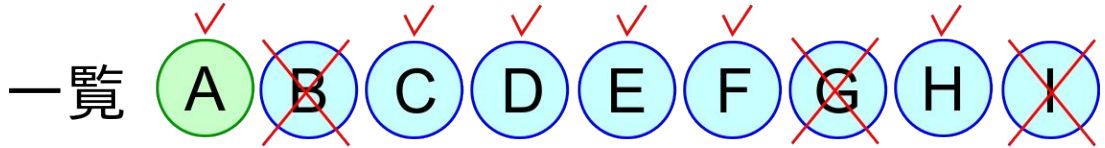


一覧 A B C D E F G H I

具体的なGCの仕組み

- 手順2：オブジェクトの一覧を先頭から見ていって、マークが付いていないオブジェクトを捨てればよい（「スイーピング」と呼ぶ）





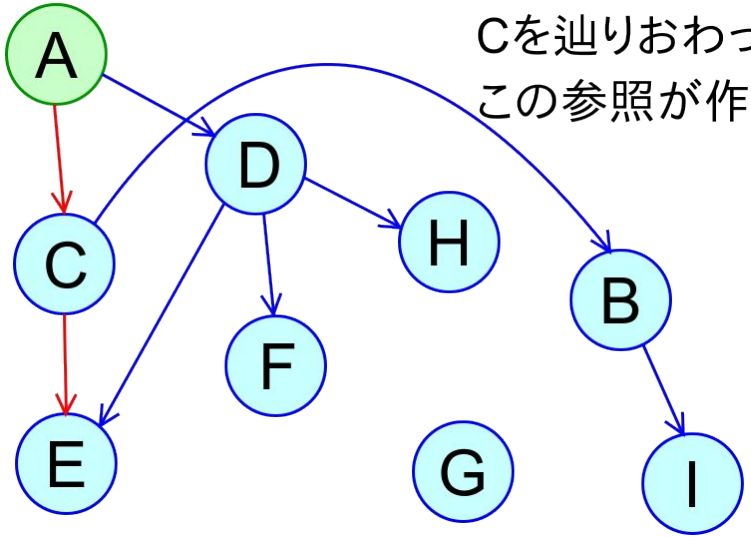
具体的なGCの仕組み

- 以上が、「マーク & スイープGC」と呼ばれるもっとも基本的なGCのやり方です

- 単純なマーク & スイープGCには（これから述べる）性能上の問題があるので、多種多様な改良型のGCが発明されています

マーク & スイープGCの問題点

- マーキングしている最中にグラフの形が変わると、捨ててはいけないオブジェクトを捨ててしまうことになります
  - もしCを辿りおわったあとで、C ⇒ Bへの参照が作られてしまうと、BとIを辿りそこねて、BとIが捨てられてしまう



Cを辿りおわったあとで、
この参照が作られたとしたら...

マーク & スイープGCの問題点

- なので、結果の正しさを保証する（＝間違って必要なオブジェクトを捨てない）ためには、マーキング中にグラフが固定されている必要があります

- つまり、GCがマーキングをしている最中にはChromeの実行を完全に停止させておく必要があります

マーク & スイープGCの問題点

- マーキングが短時間で終われば問題はないのですが、最近のWebページが使用するデータ量は爆発的に増えてきていて、グラフを辿るのに数百ミリ秒～数秒かかってしまうことも珍しくありません
  - そうなると、Chromeが数百ミリ秒～数秒停止してしまいます

マーク & スイープGCの問題点

- これこそが、ひと昔前のChromeでカクカクが起きていた原因です
  1. Google Mapsを指でグイーーーンと動かすと、地図が動くにつれて大量のデータが作られ、大量のデータが不要になる
  2. 不要なデータを捨てるために、定期的にGCが走る
  3. データ量が多いので、GCがグラフを辿るのに時間がかかる
  4. その間、Chromeの実行が止まる...（=> カクカク）

賢いGCの必要性

- Chrome（にかぎらずソフトウェア）をサクサク動かすためには、賢いGCを作り込むことが必須技術になります
  - Webページで使われるデータ量が増えるにつれて、この問題はどんどん深刻になってきていて、さまざまな改良型GCが発明されています

賢いGCの必要性

- どれくらい賢ければいいかというと、60 frames/secがひとつの目安と言われています
  - 1秒間に60回の速度でページが再描画されれば、人間にとってWebページの操作がスムーズに感じられます
  - 言い換えると、GCが16ミリ秒以上Chromeの実行を止めてしまうと、人間が気づくようなカクカクが起きます
  - 「GCは16ミリ秒以上停止してはいけない」

16ミリ秒の壁

- 「GCは16ミリ秒停止してはいけない」
  - GCの実行時間はオブジェクトの量に比例します
  - だからといって、「Webページのオブジェクト量が多かったので、50ミリ秒止まってしまいました。ごめんwwwww」というのは許されません
    - 最近のWebページのオブジェクト量は巨大

16ミリ秒の壁

- 「GCは16ミリ秒停止してはいけない」
  - 要するに、「16ミリ秒では辿れない量のオブジェクトを作るWebページに対しても、16ミリ秒以下の停止時間で動くGC」を考える必要があります
  - どうやったらそんなことができるのか？

ここまでのまとめ

- マーク & スイープGCでは、マーキングをやっている最中にオブジェクトグラフが固定されている必要があります
  - マーキング中にはChromeの実行を停止する必要がある
  - これがカクカクの原因

- スムーズなChromeを作るためには、「Webページのオブジェクト量がいくら巨大であろうとも、16ミリ秒以上止まらない」賢いGCを作る必要があります

16ミリ秒の壁をどう越えるか

こういう遊びをやったことはないですか？

- ゴミ箱をひっくり返して、「ゴミの寿命」の分布状況を調べる遊び
  - ゴミの寿命 ＝ その物体を使い始めてから捨てるまでにかかった時間

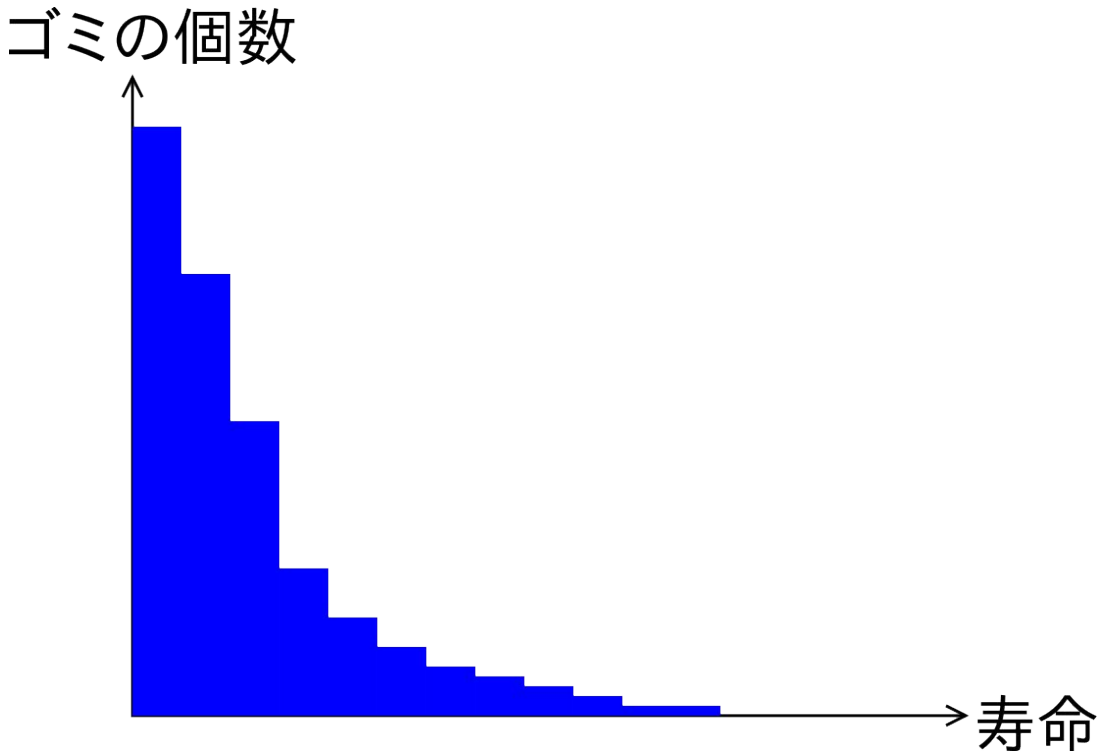
こういう遊びをやったことはないですか？

- ゴミ箱をひっくり返して、「ゴミの寿命」の分布状況を調べる遊び
  - ティッシュ：16枚（ティッシュの寿命＝10秒）
  - お菓子袋：2枚（お菓子袋の寿命＝5分）
  - 服：1枚（服の寿命＝1年）

- 直観的にわかること：
  - ゴミの大部分は寿命の短い物体である
  - 寿命の長い物体はめったにゴミにならない

こういう遊びをやったことはないですか？

- ゴミ箱をひっくり返して、「ゴミの寿命」の分布状況を調べる遊び



社会科学的考察

- ゴミ箱からわかること:
  - 「ゴミの大部分は寿命の短い物体である」
  - ほぼ同義なこととして、「新しい物体の死亡率は高く、古い物体の死亡率は低い」
    - ティッシュはすぐにゴミになる確率が高い
    - 服はすぐにゴミになる確率は低い

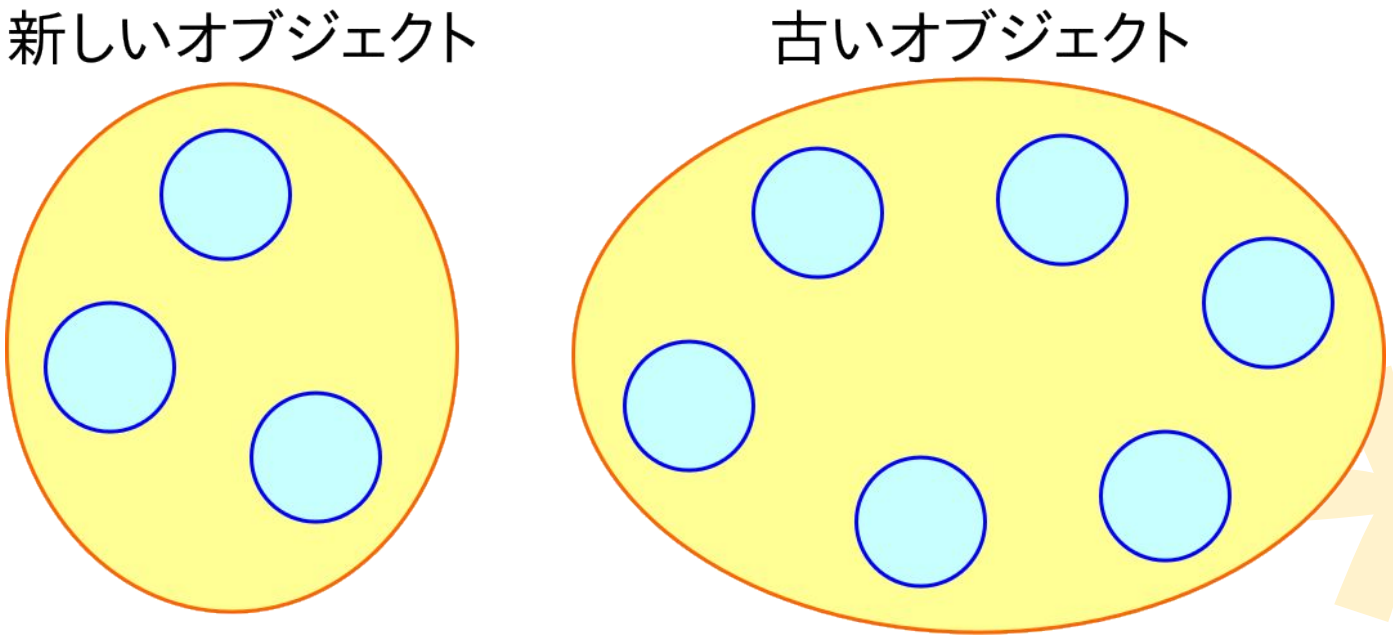
- 同じことは動物の世界にもあてはまります
  - 乳幼児死亡率は高く、成人死亡率は低い

社会科学的考察

- この傾向を利用したら、賢いGCを作れないか？
  - 世代別GCのアイディア

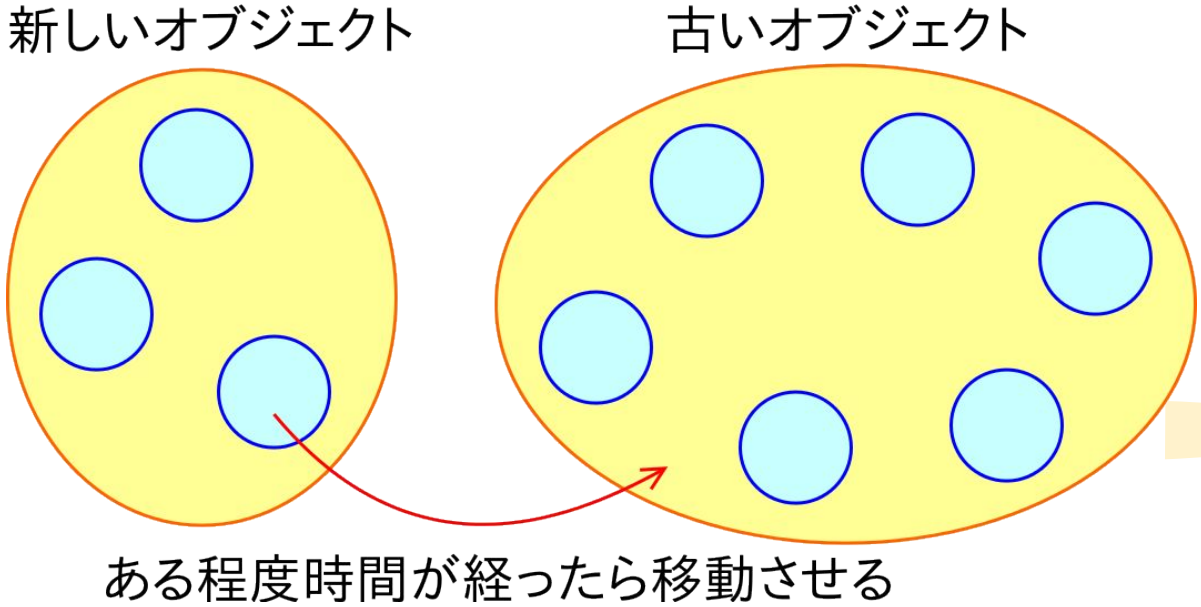
世代別GCのアイディア

- オブジェクト全体を、「新しいオブジェクト」と「古いオブジェクト」に分けて管理しておく



世代別GCのアイディア

- 具体的には、新しく作られたオブジェクトは「新しいオブジェクト」のグループに入れて、しばらく経ってもまだ生き残っているオブジェクトを「古いオブジェクト」のグループに移動させる



世代別GCのアイディア

- 先ほどの考察でわかったことは、「新しいオブジェクト」の死亡率は高く、「古いオブジェクト」の死亡率は低い

- ということは、「新しいオブジェクト」に対してだけ重点的にGCをかけたら効率的なのでは？
  - 10回中9回のGCは、「新しいオブジェクト」に対してだけ走らせる
    - この「限定的な」GCで大部分のゴミを回収できるはず！
  - 10回中1回のGCは、オブジェクト全体に対して走らせる

世代別GCがうまく動くための条件

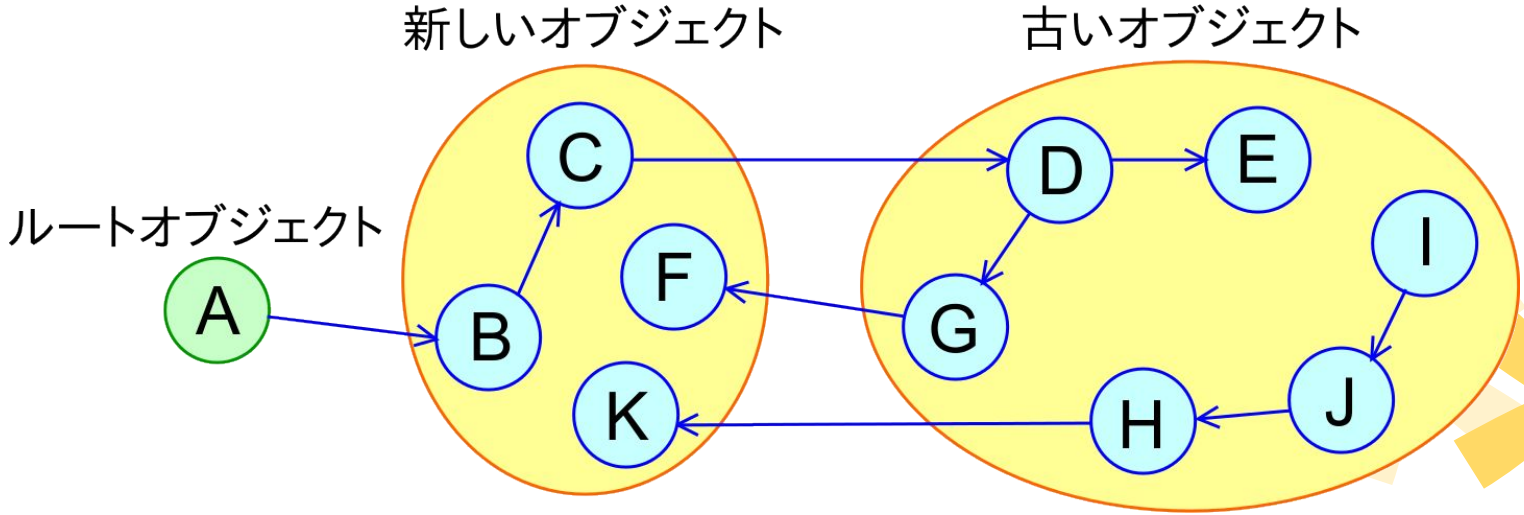
- 条件1:「新しいオブジェクト」の死亡率が「古いオブジェクト」の死亡率よりも十分に高いこと
  - 現実的な傾向として成り立っていることが多い
- 条件2:「新しいオブジェクト」の個数が「古いオブジェクト」の個数よりも十分に少ないこと
  - これも現実的には成り立っていることが多い
  - あなたが持っている資産全体(服、家具、家電 etc)のうち、今日手に入れた資産(ティッシュ、菓子袋 etc)の割合はずっと少ないですよね？

世代別GCのインパクト

- この2つの条件が成り立っているとき、「新しいオブジェクト」に対して重点的にGCをかけることで効率的にゴミを回収できるわけです
  - 10回中9回のGCは速い（16ミリ秒以下に収まる）
    - しかもこれだけで大部分のゴミは回収できる
  - 10回中1回のGCは遅い
    - これの頻度が十分低ければOK

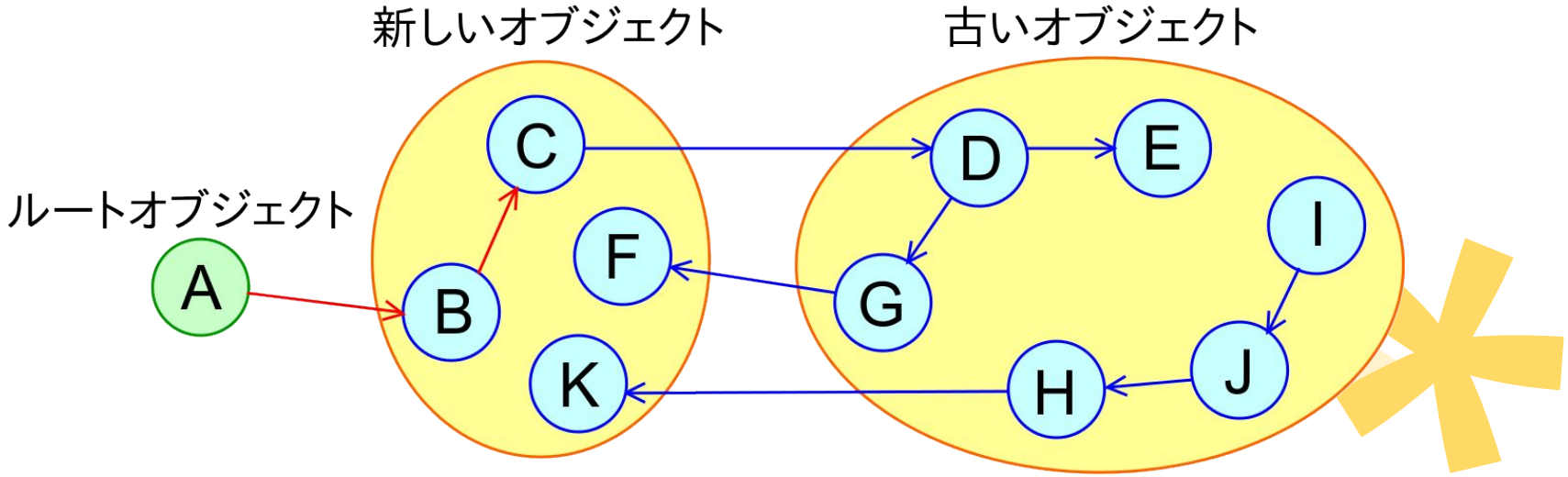
とはいえ、世代別 GCを作るのは簡単じゃない

- ところで、「新しいオブジェクト」に対してだけGCをかけるにはどうすればいいでしょうか？
  - グラフは「新しいオブジェクト」と「古いオブジェクト」にまたがっている…



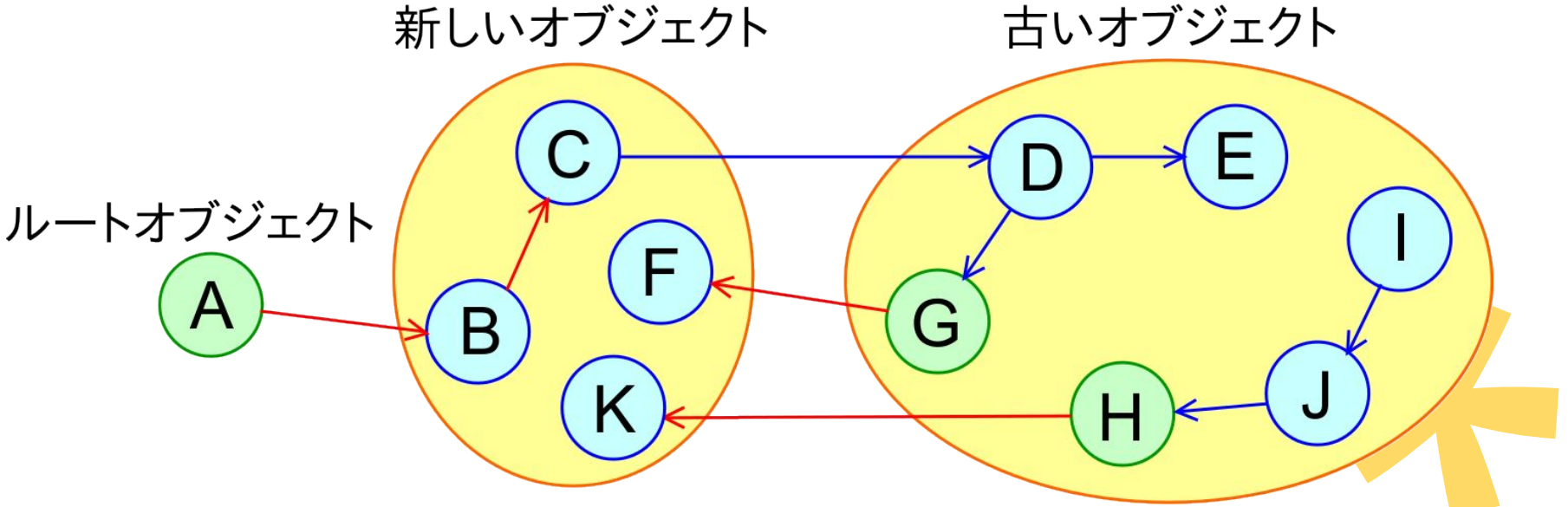
とはいえ、世代別 GCを作るのは簡単じゃない

- 単純に「新しいオブジェクト」の部分のグラフしか辿らないことにすると、回収してはいけないオブジェクトを回収してしまいます
  - もしCで辿るのをやめてしまうと、Fを回収してしまう
  - かといって、「古いオブジェクト」の部分も辿ったら本末転倒



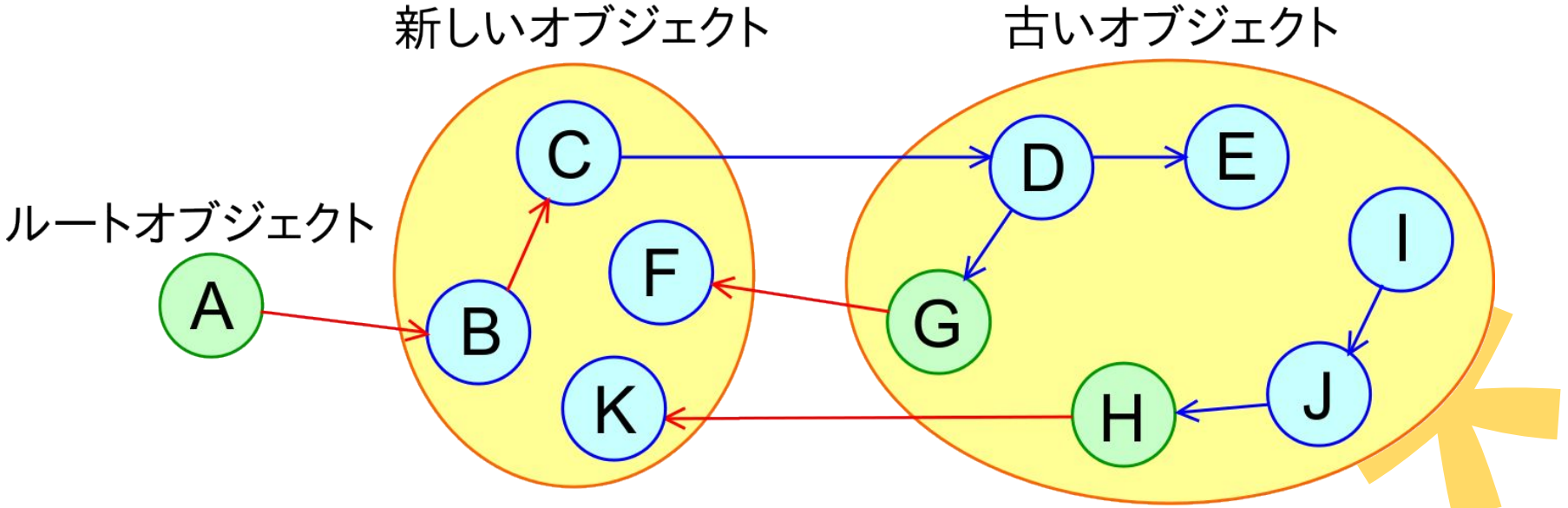
とはいえ、世代別 GCを作るのは簡単じゃない

- この問題を解決するためには、「新しいオブジェクト」を指している「古いオブジェクト」もルートオブジェクトだとみなして、グラフを辿ることにすればOKです



とはいえ、世代別 GCを作るのは簡単じゃない

- この場合、本来回収されるべきオブジェクトが回収されないことが起きえますが（＝GCの効率が悪くなる）、これはあきらめます
  - 本来Kは回収されるべきだが、この方法では回収できない



ここまでのまとめ

- 解きたい問題:「どうすれば16ミリ秒以上止まらないGCを作れるか？」

- 世代別GCのアイディア:「新しいオブジェクト」に対してだけ重点的にGCをかけることで、大部分のGCの停止時間を短くする

- 詳細は省略しますが、残りのGCの停止時間も短くする技術としてインクリメンタルGCというのがあって、Chromeではこれらの最先端のGC技術を駆使しています

最後に

なぜそこまで性能が重要なのか

- Chromeの性能チームは毎日1ミリ秒のレベルで戦っています

- なぜそこまでして性能が重要なのか？
  - 「別にいまのChromeって、十分速い気がするんですけど・・・」
  - 「そんな1ミリ秒のレベルで汗を流さなくても・・・」

なぜそこまで性能が重要なのか

いや、重要なんです

なぜそこまで性能が重要なのか

- 「人々が現状の速度に不満を感じてないから、これ以上速くしなくてもよい」というのは違います

- 起こすべき技術の流れ:
  - とにかく速くする ⇒ Webでできることがもっと増える
  - 実際問題として、5年前のChromeでは現在のGoogle Mapsを快適には動かせませんでした
  - Chromeに性能上の進化があったからこそ、動かせるWebアプリの数や質が格段に増えたわけです

なぜそこまで性能が重要なのか

- 「GCの停止時間で16ミリ秒の壁を越える」という話だけ聞くと、何のためにそんなことが必要になるのかわからないですが、Chromeがサクサク動く背景・Webが進化してきた背景には、こういうコンピュータ・サイエンスの地道な積み重ねがあるんです

最後に

- 今日は「Chromeのなかのコンピュータ・サイエンス」ということでGCの話を掘り下げてみました

- 直接ユーザの目に触れるプロダクトを作ったり、新しい機能をローンチしたりすることだけがエンジニアリングではありません

- プロダクトの基盤にはコンピュータ・サイエンスの深い世界が広がっていて、その進化がプロダクトの進化を支えています！