



# А что нового в GC в .NET 10?



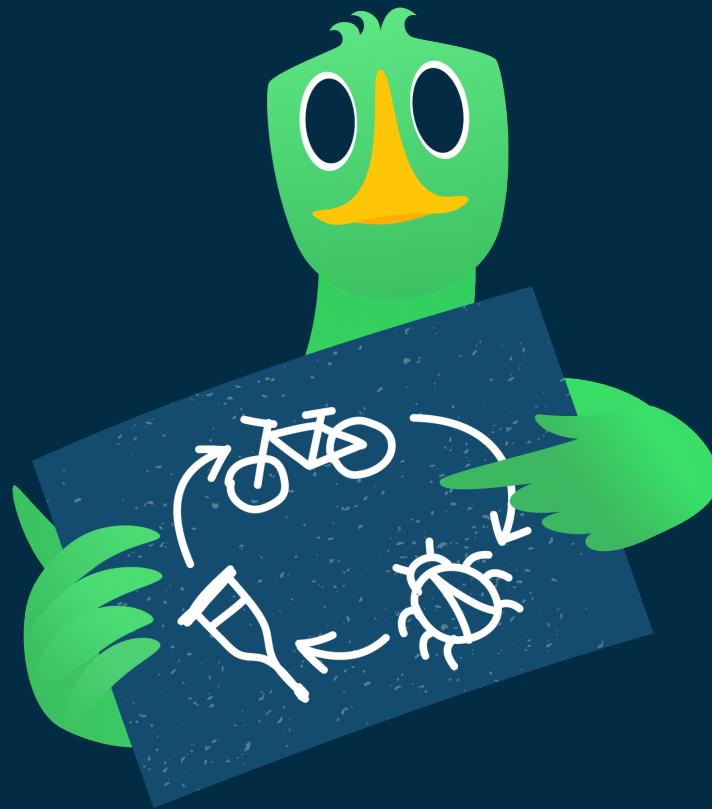
**Антон Воронцов**

**Руководитель команды платежных интеграций**  
**Ozon Bank**

# GC

# GC JIT

GC  JIT

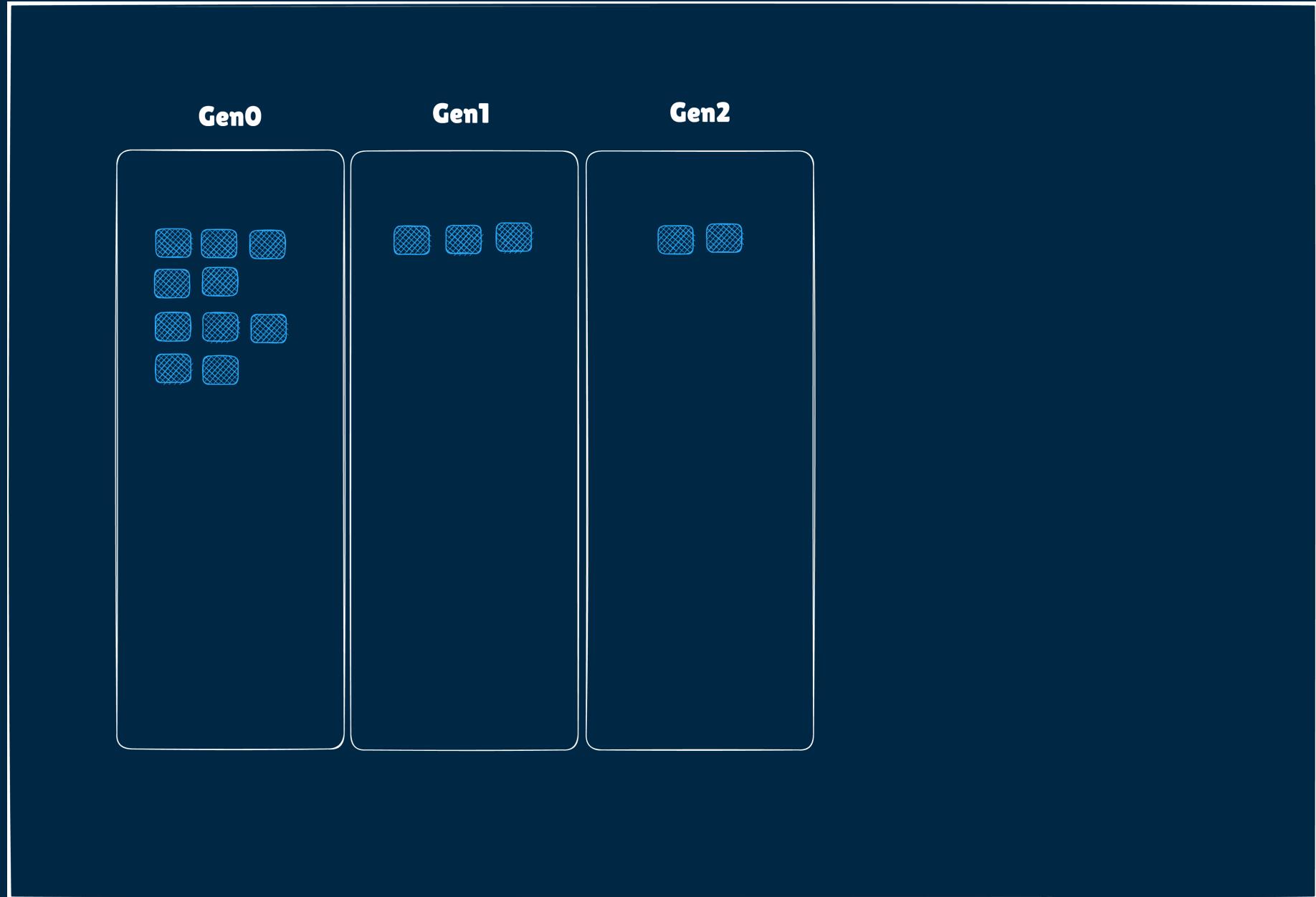


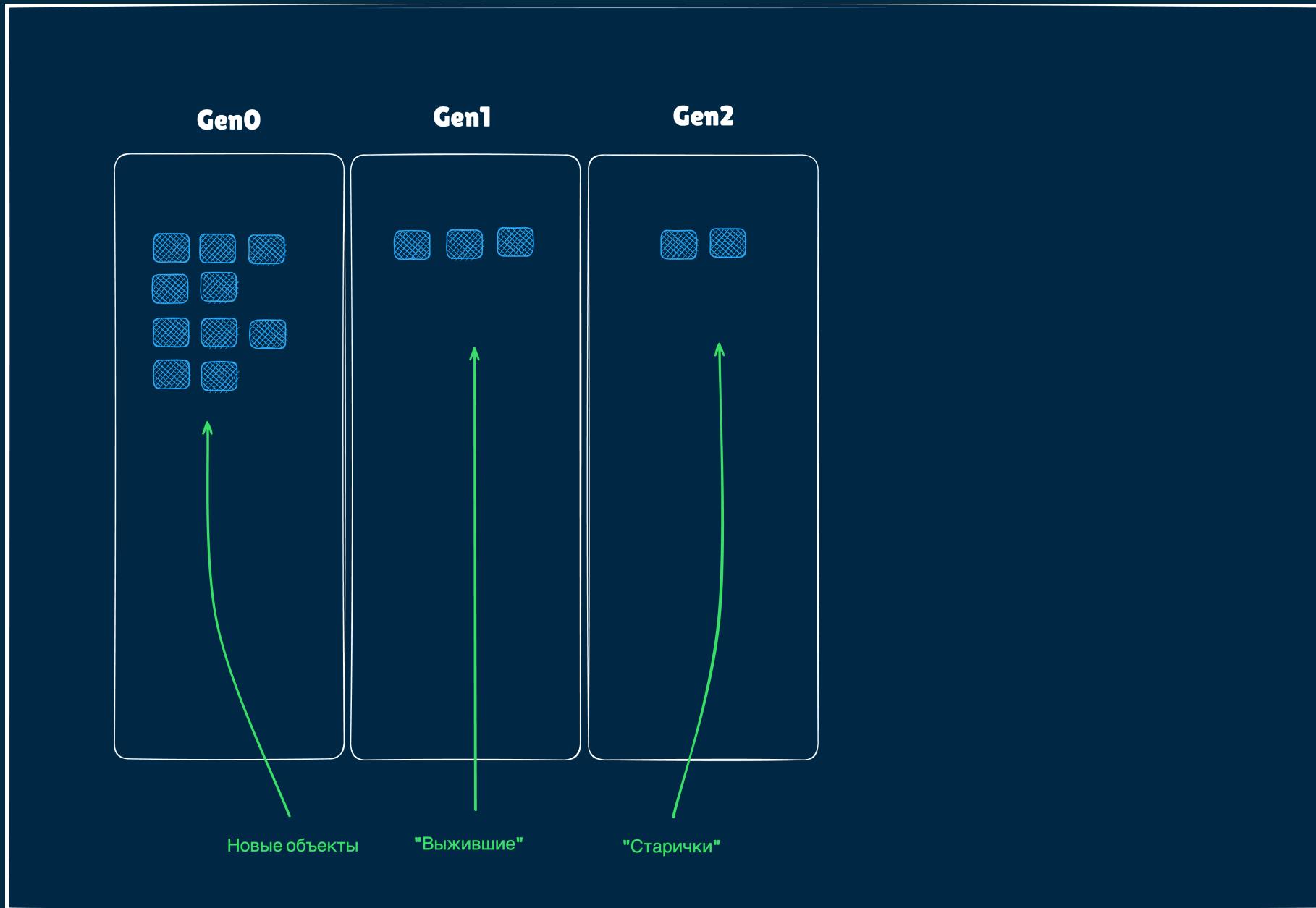
**Спойлер**  
**Прорывов нет, но есть**  
**«улучшмы»**

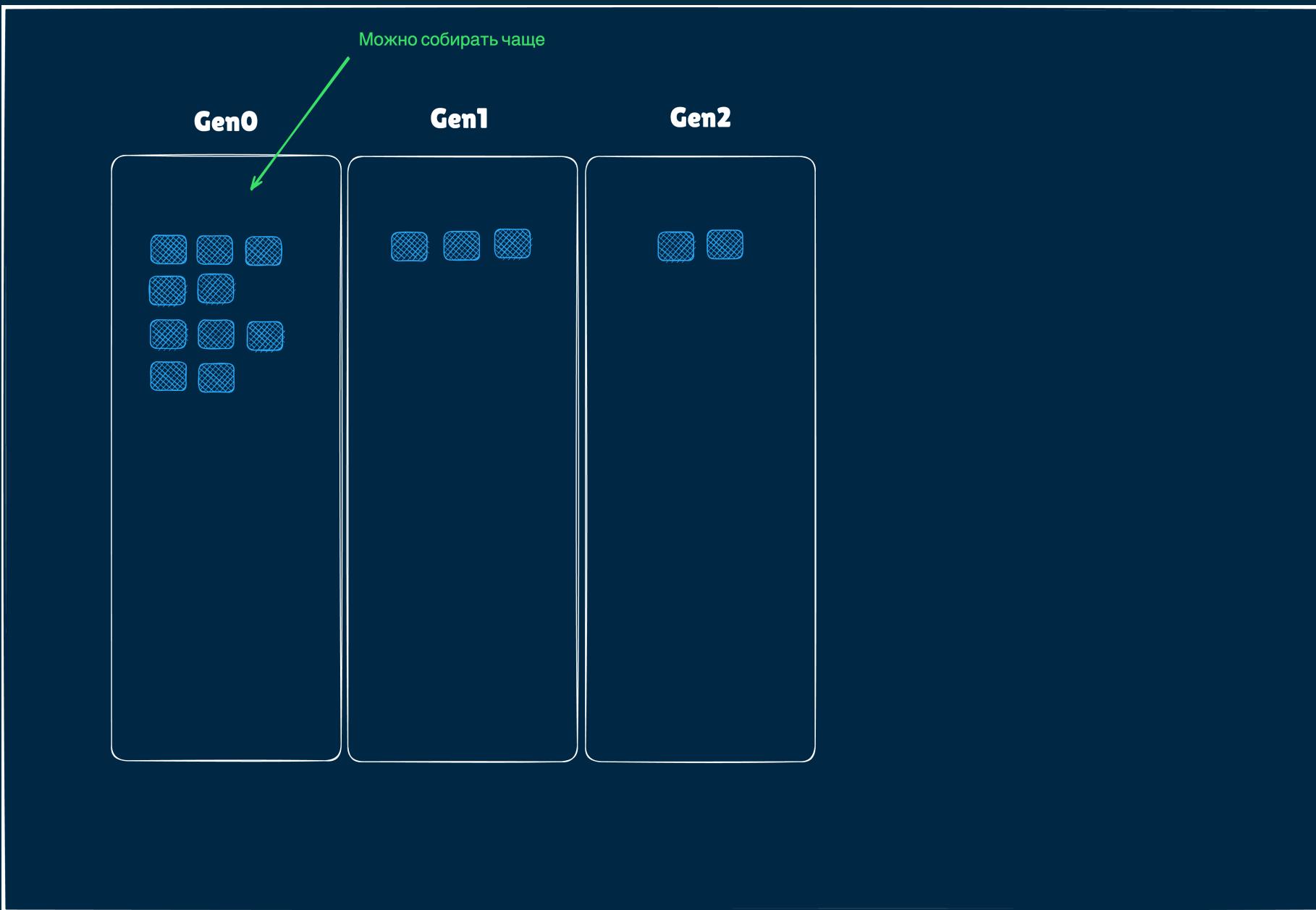


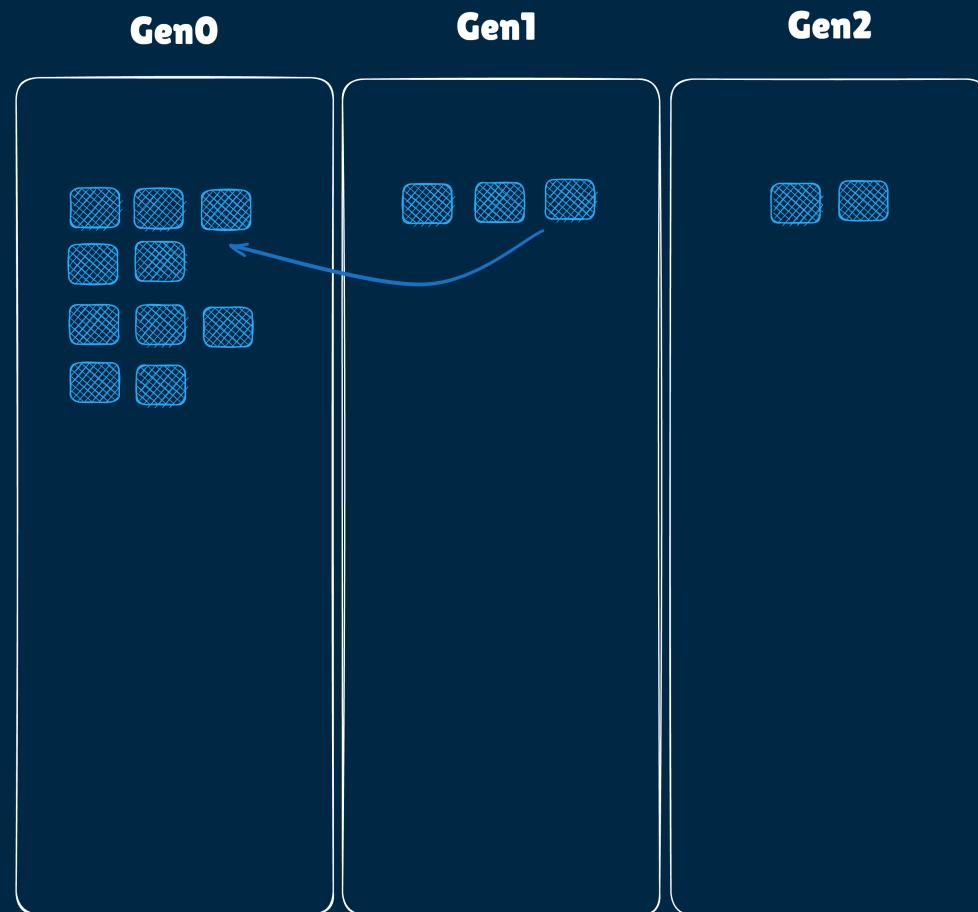
# Write Barriers

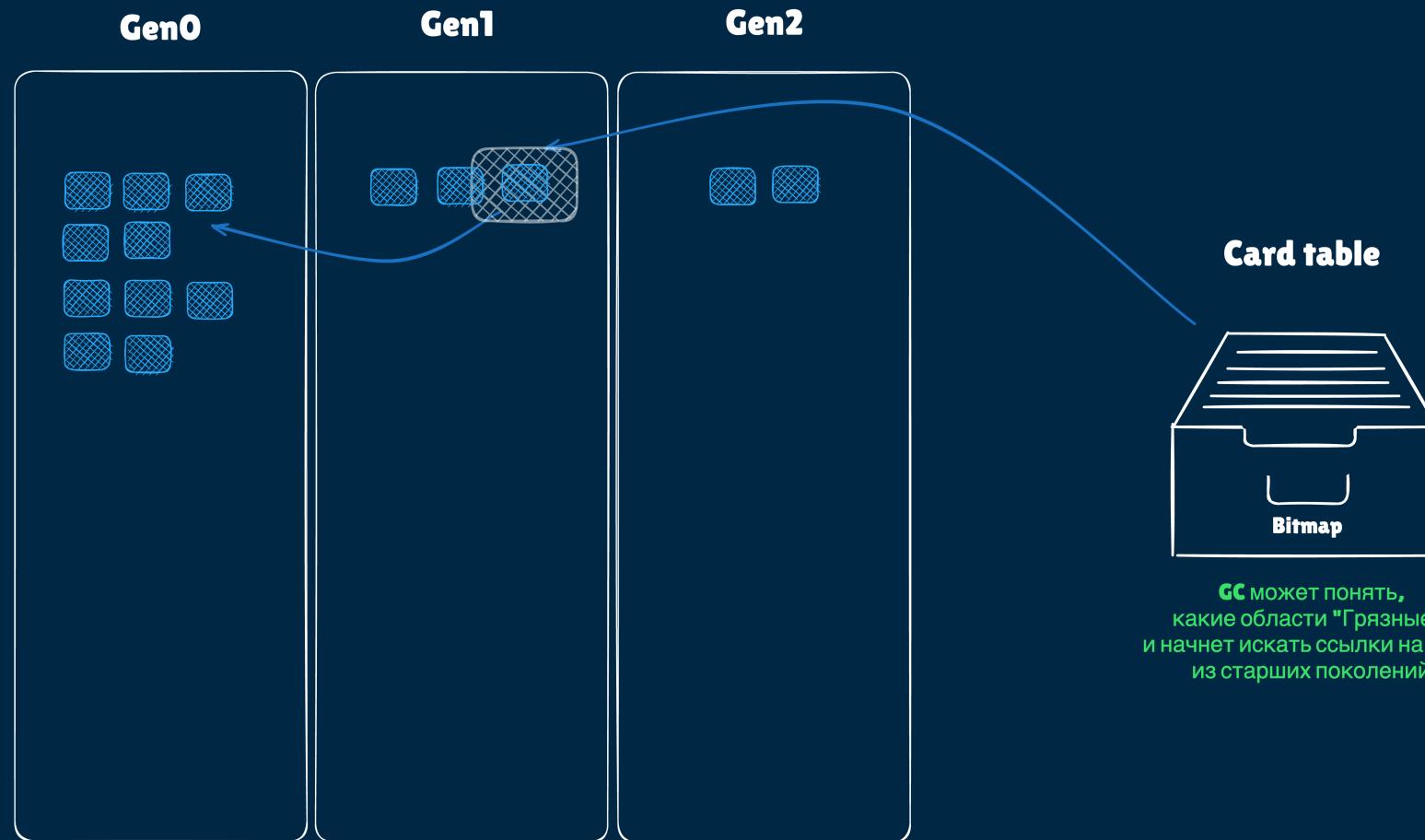


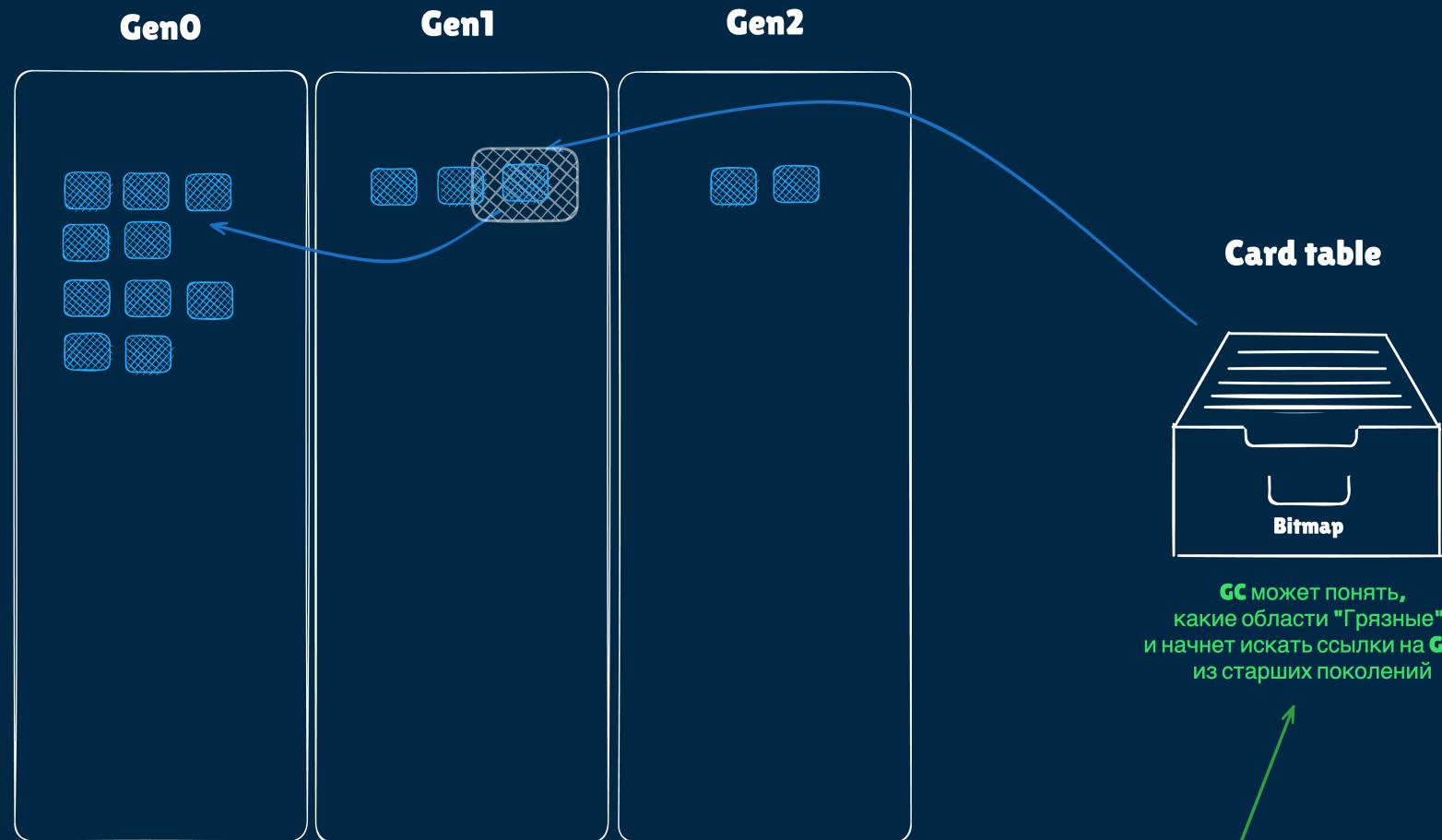












Для этого нужен **Write Barrier**

Меньше Write Barriers  
означает меньше  
работы для GC



# Что поменялось в .NET 10?



# Что поменялось в .NET 10?

- Больше нет WB для ref struct



# Что поменялось в .NET 10?

- Больше нет WB для ref struct
- Return buffers с аллокацией на стеке



# Что поменялось в .NET 10?

- Больше нет WB для ref struct
- Return buffers с аллокацией на стеке
- Переиспользованный код  
*WriteBarrierManager* для arm64  
(tradeoff, потому что WB стали «дороже», но их стало меньше)



# Примеры кода



```
1 public ref struct MyRefStruct
2 {
3     public object Obj1;
4     public object Obj2;
5     public object Obj3;
6 }
7 // ⚡ При обновлении ссылок больше нет Write Barriers
8
9
10 public record struct Person(
11     string FirstName,
12     string LastName,
13     string Address,
14     string City);
15
16 public Person GetPerson() => new(_firstName, _lastName, _address, _city);
17 // ⚡ А здесь больше нет Write Barriers для Return Buffer
```



# Dynamic Adaptation To Application Sizes (DATAS)



# Что за зверь такой, этот DATAS?



# Что за зверь такой, этот DATAS?

- Механизм GC, который динамически подстраивает размеры поколений, количество куч и частоту сборок



# Что за зверь такой, этот DATAS?

- Механизм GC, который динамически подстраивает размеры поколений, количество куч и частоту сборок
- Основан на *runtime-метриках*, а не конфигурации



# Что за зверь такой, этот DATAS?

- Механизм GC, который динамически подстраивает размеры поколений, количество куч и частоту сборок
- Основан на *runtime-метриках*, а не конфигурации
- GC смотрит на профиль работы приложения, а не статичный конфиг



# Что за зверь такой, этот DATAS?

- Механизм GC, который динамически подстраивает размеры поколений, количество куч и частоту сборок
- Основан на *runtime-метриках*, а не конфигурации
- GC смотрит на профиль работы приложения, а не статичный конфиг
- **Механизм не новый.** Появился в .NET 8. Включен по умолчанию в .NET 9



# А раньше было как?



# А раньше было как?

- Эвристики:
  - CPU count (== heap count)
  - Частоту сборок



# А раньше было как?

- Эвристики:
  - CPU count (== heap count)
  - Частоту сборок
- Память выделялась, но не использовалась



# А раньше было как?

- Эвристики:
  - CPU count (== heap count)
  - Частоту сборок
- Память выделялась, но не использовалась
- Медленная адаптация под изменения

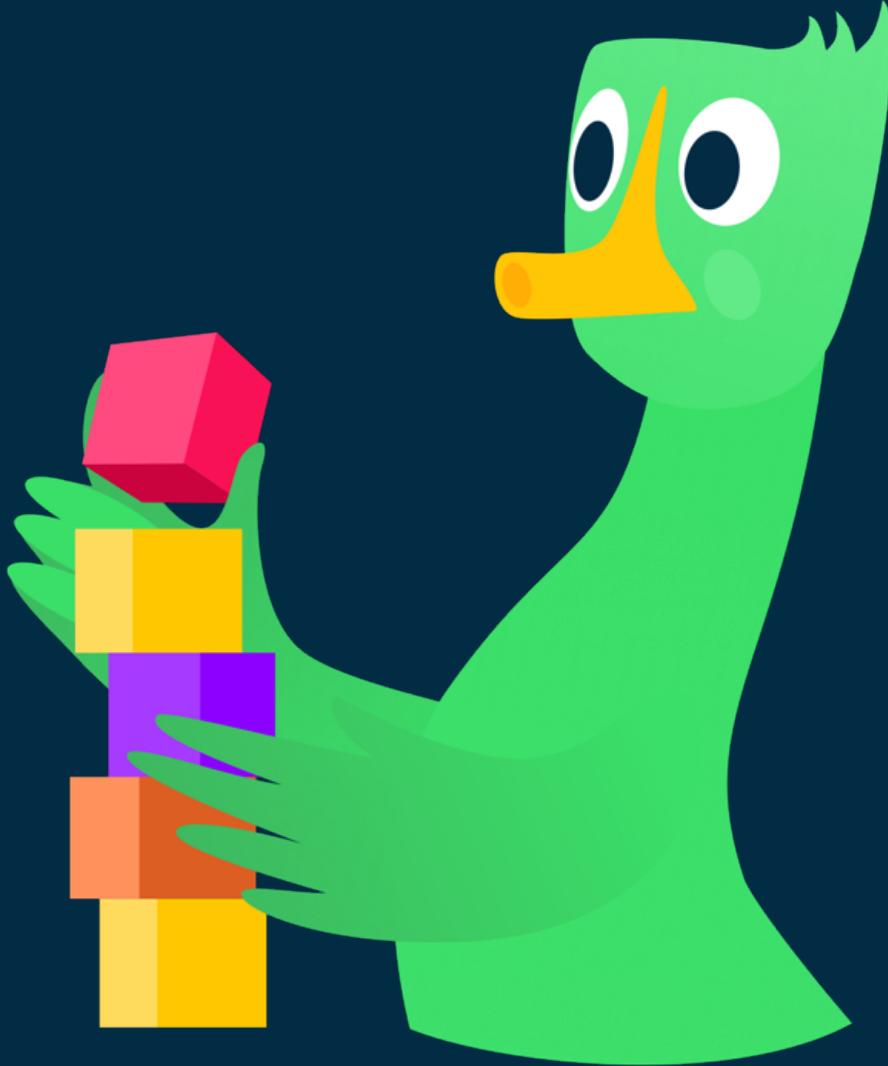


# А раньше было как?

- Эвристики:
  - CPU count (== heap count)
  - Частоту сборок
- Память выделялась, но не использовалась
- Медленная адаптация под изменения
- GC несвоевременный, Gen0 маленький

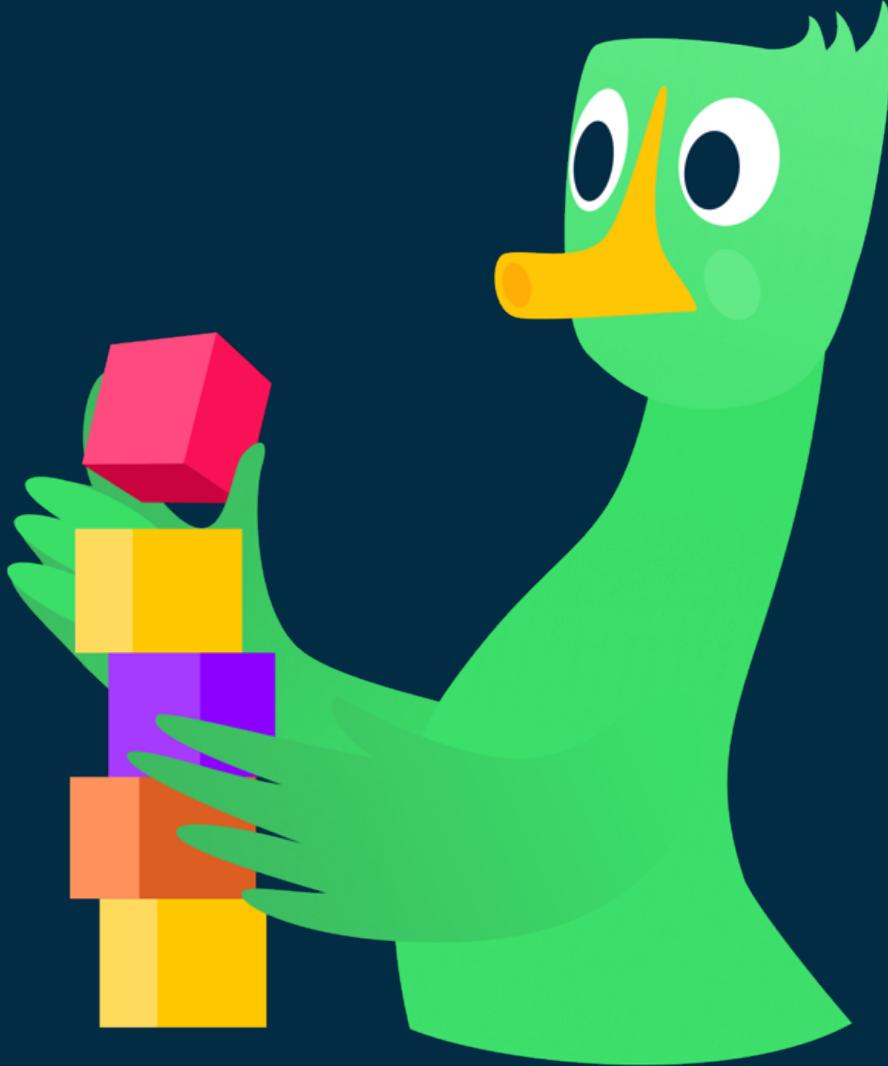


# Как DATAS работает



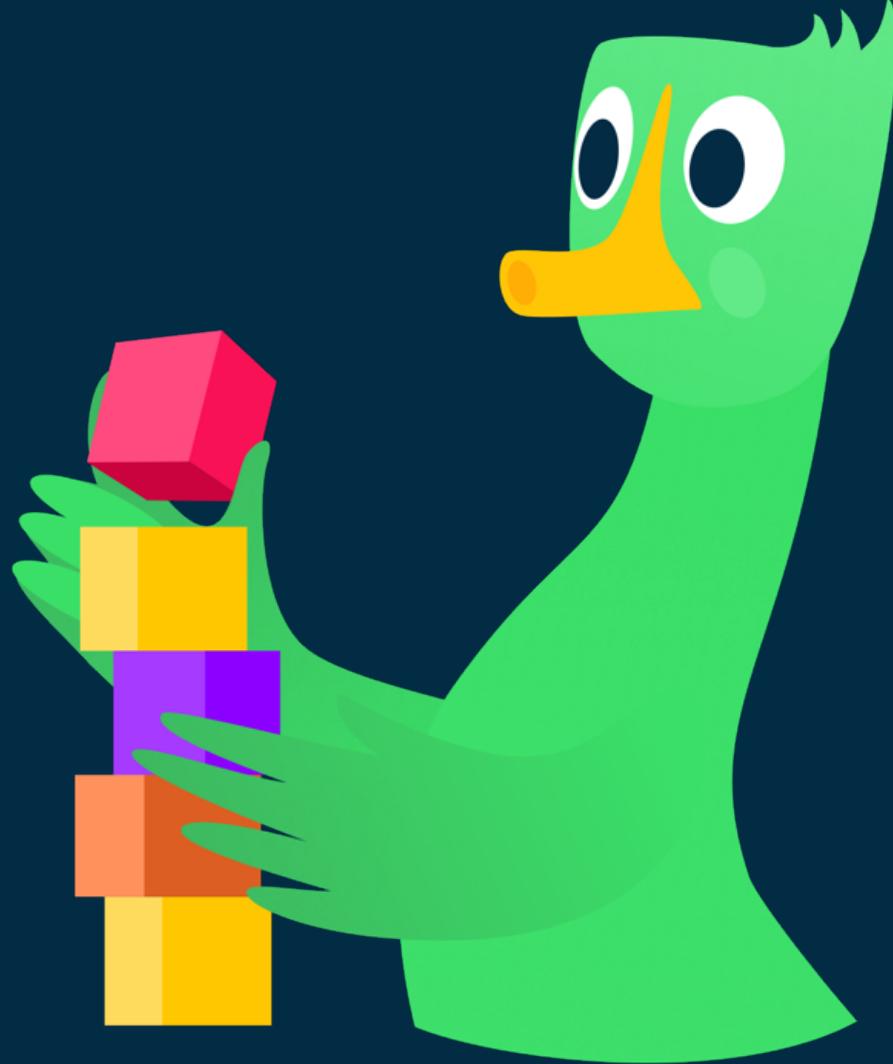
# Как DATAS работает

- *Что анализируются:*



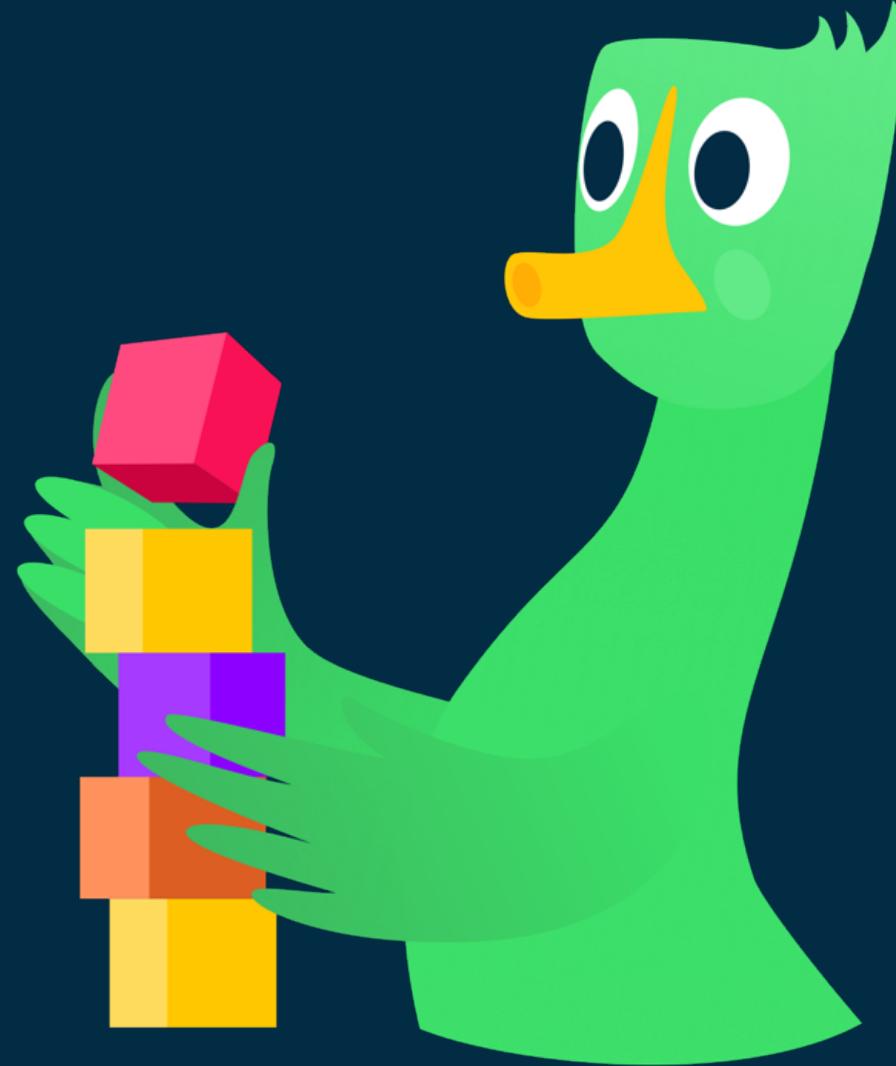
# Как DATAS работает

- *Что анализируются:*
  - Скорость аллокаций
  - Survival rate объектов

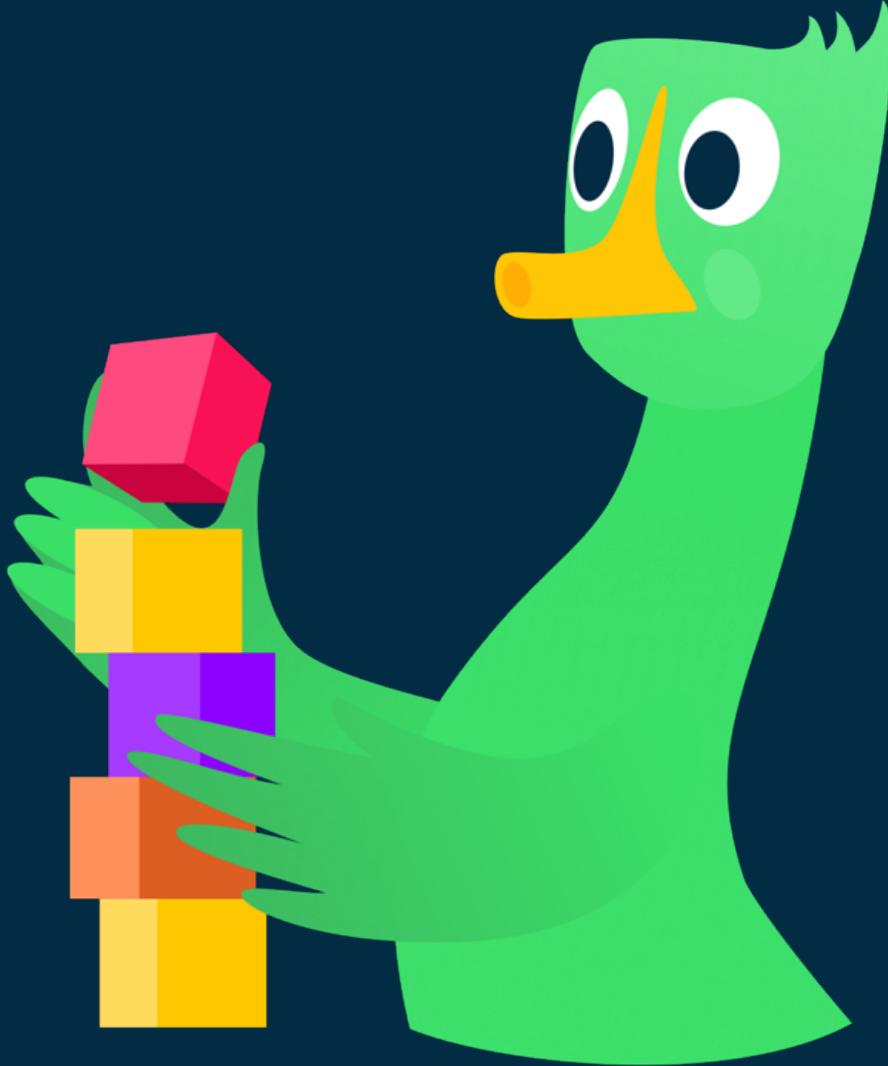


# Как DATAS работает

- *Что анализируются:*
  - Скорость аллокаций
  - Survival rate объектов
  - Частота GC
  - Эффективность прошлых сборок
  - Фрагментация heap

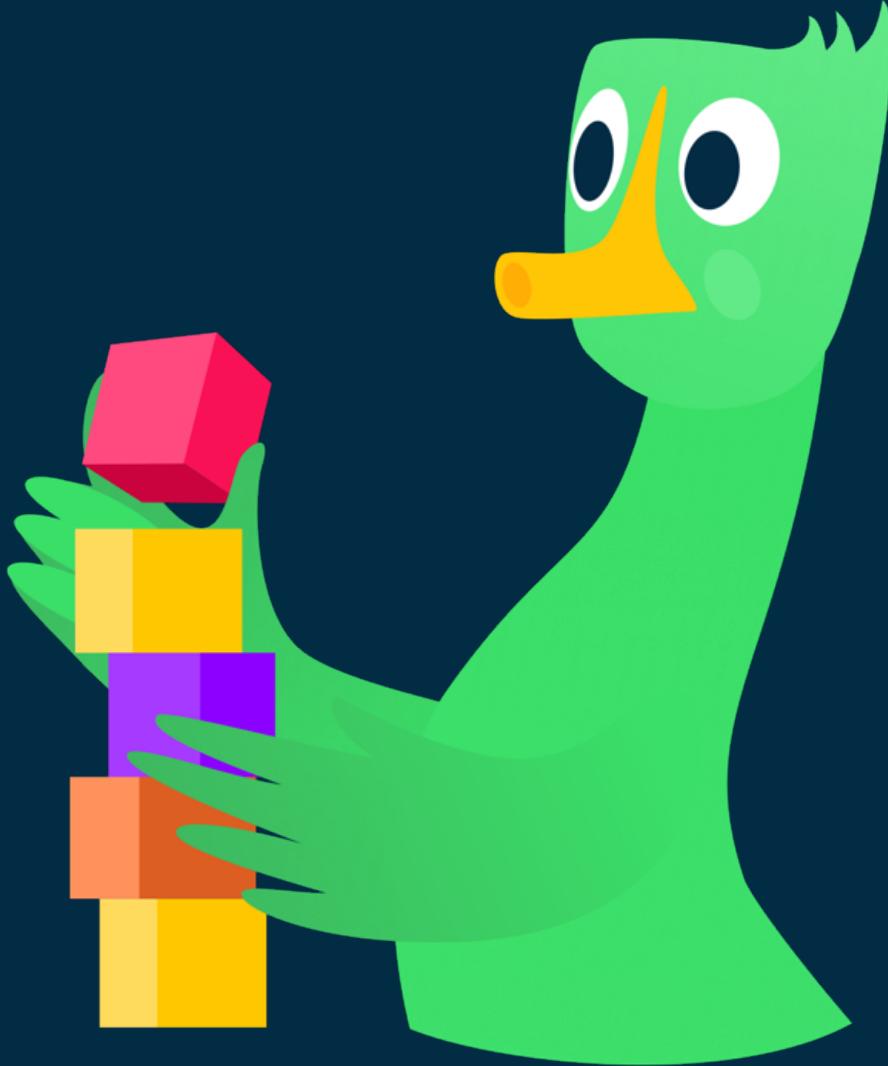


# Как DATAS работает



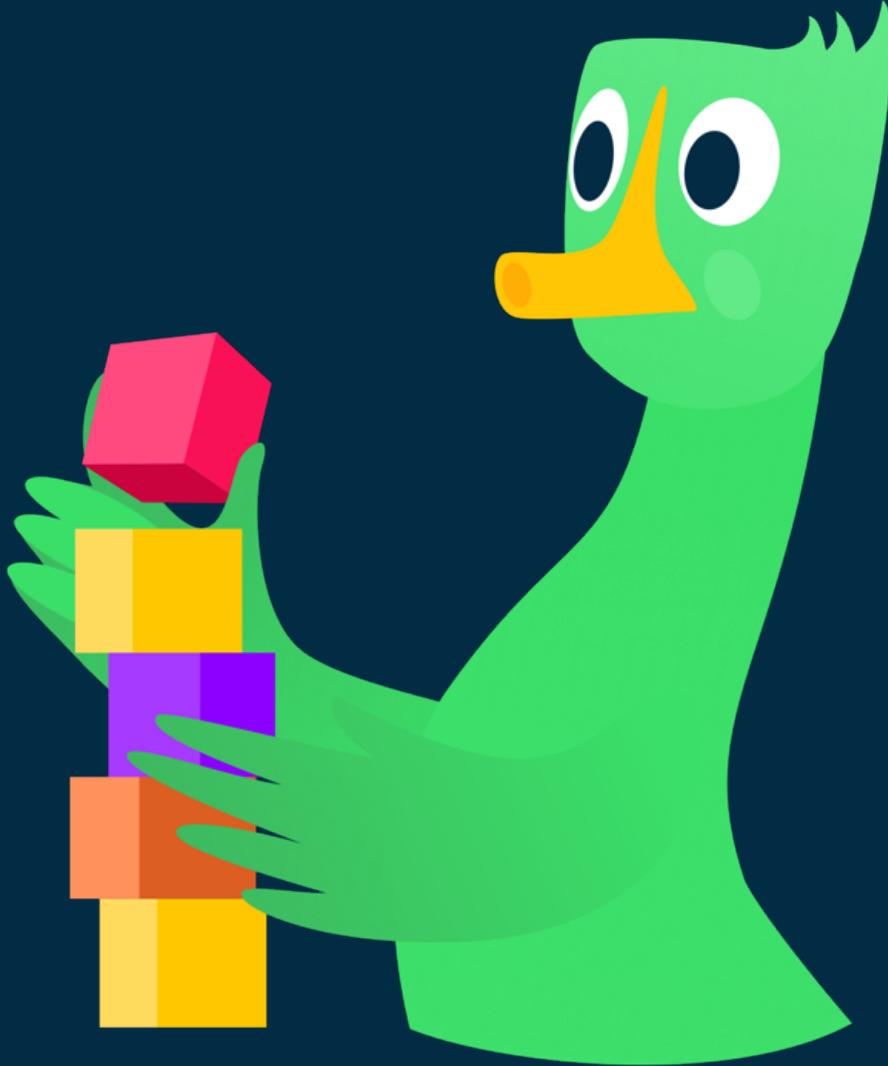
# Как DATAS работает

- *Что GC делает:*



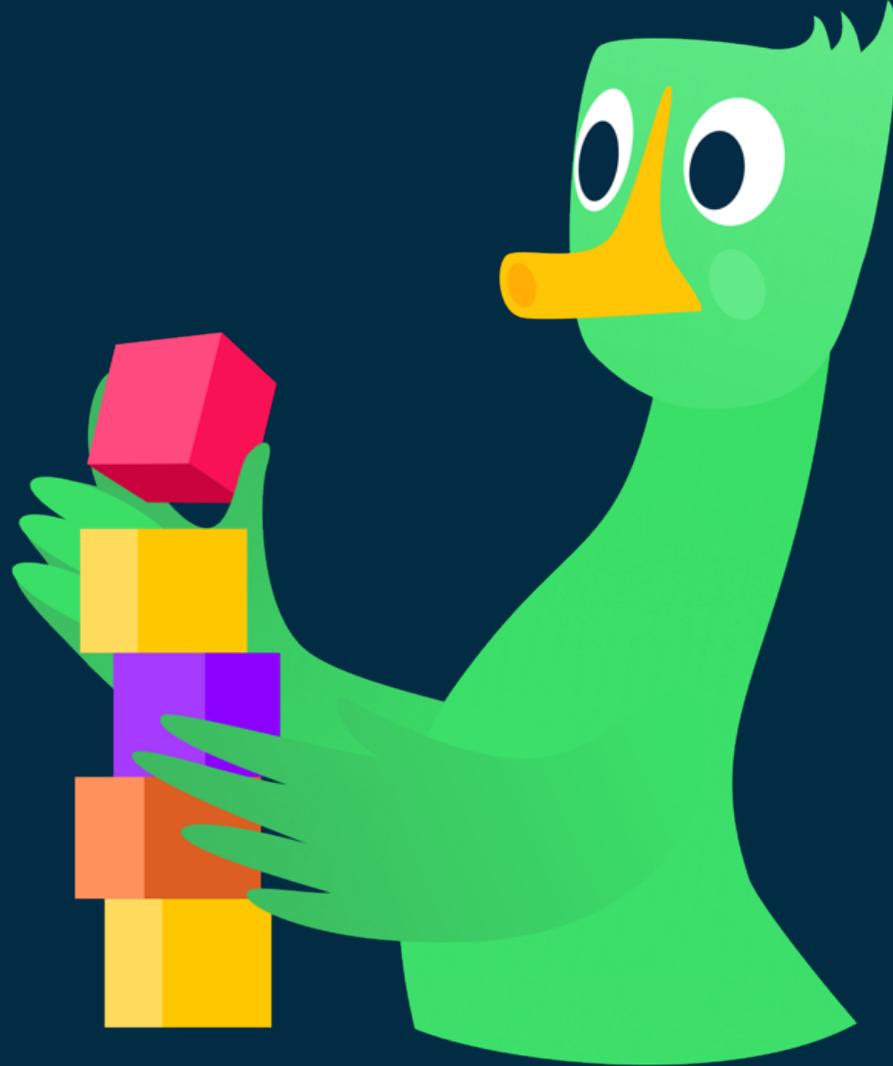
# Как DATAS работает

- *Что GC делает:*
  - Увеличивает или уменьшает размеры поколений



# Как DATAS работает

- *Что GC делает:*
  - Увеличивает или уменьшает размеры поколений
  - Тем самым сглаживает рост heap



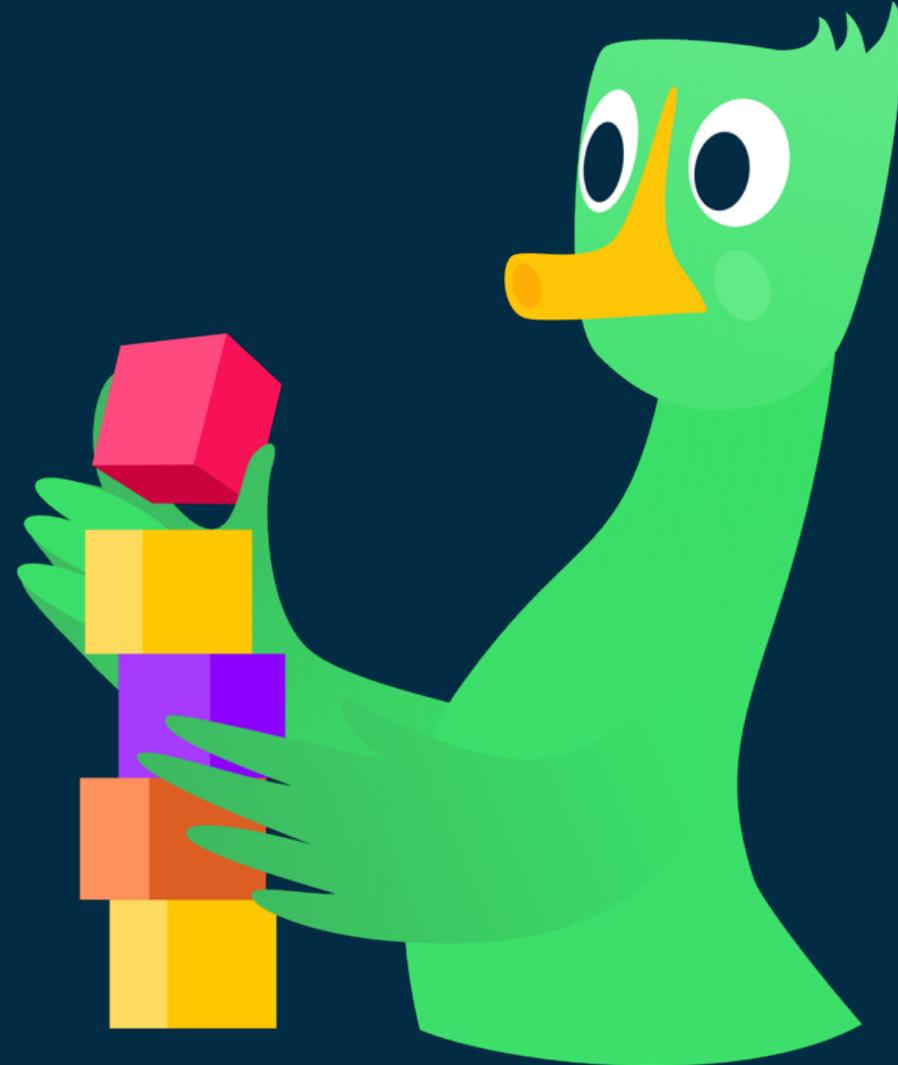
# Как DATA\$ работает

- *Что GC делает:*
  - Увеличивает или уменьшает размеры поколений
  - Тем самым сглаживает рост heap
  - Количество heap тоже оптимизируется (Server GC), не держится память впустую



# Как DATA\$ работает

- *Что GC делает:*
  - Увеличивает или уменьшает размеры поколений
  - Тем самым сглаживает рост heap
  - Количество heap тоже оптимизируется (Server GC), не держится память впустую
  - Откладывает ненужные сборки



# Улучшения в .NET 10



# Улучшения в .NET 10

- Более точный рост Gen0



# Улучшения в .NET 10

- Более точный рост Gen0
- Меньше лишних Gen1



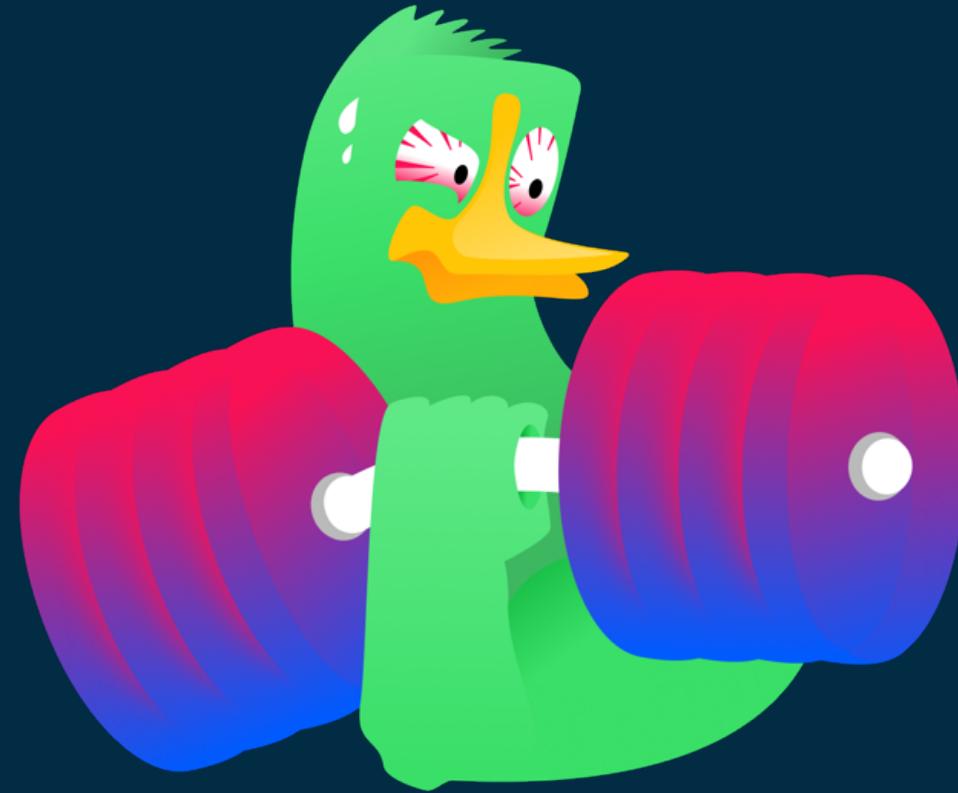
# Улучшения в .NET 10

- Более точный рост Gen0
- Меньше лишних Gen1
- Лучший учёт фрагментации



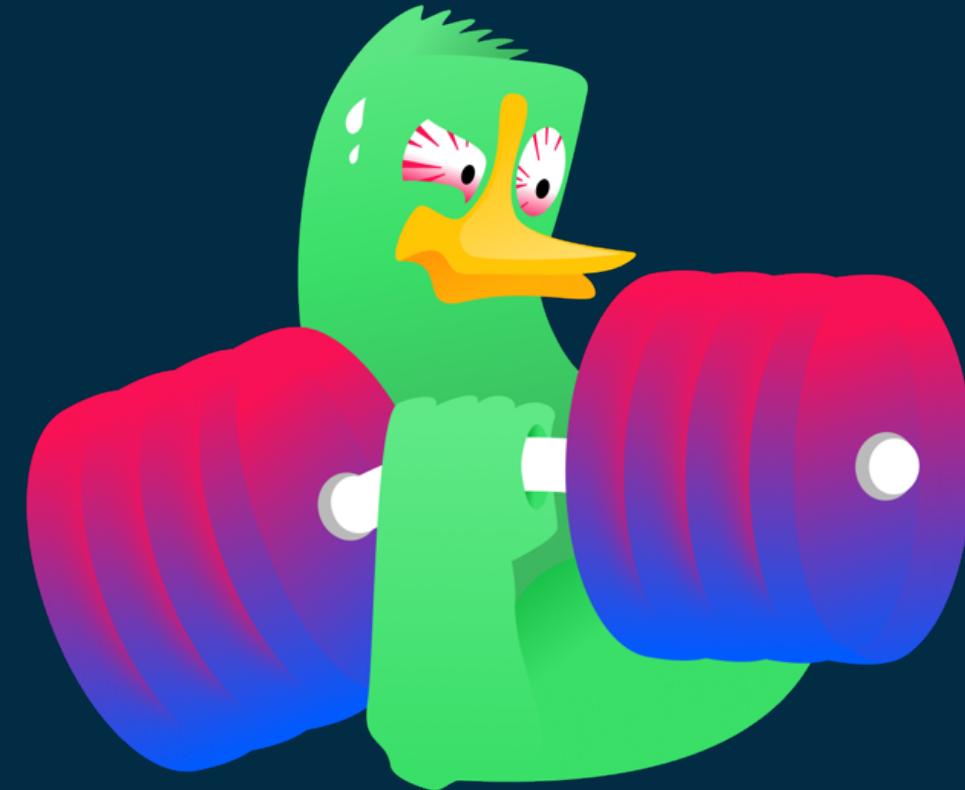
# Улучшения в .NET 10

- Более точный рост Gen0
- Меньше лишних Gen1
- Лучший учёт фрагментации
- Сглаживание GC-пауз (стоит проверить на практике)



# Улучшения в .NET 10

- Более точный рост Gen0
- Меньше лишних Gen1
- Лучший учёт фрагментации
- Сглаживание GC-пауз (стоит проверить на практике)
- Добавили "крутилки" для тонкой настройки поведения (контроль роста Gen0)



# Практический эффект



# Практический эффект

- Выше пропускная способность



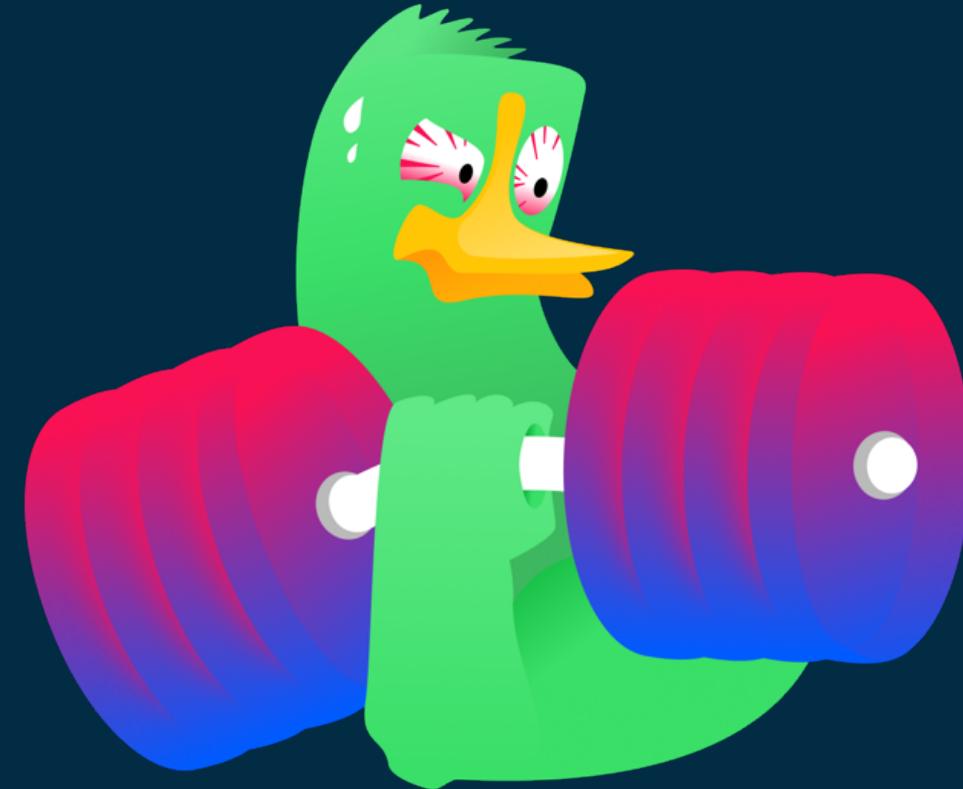
# Практический эффект

- Выше пропускная способность
- Меньше задержки



# Практический эффект

- Выше пропускная способность
- Меньше задержки
- Можно заметить эффекты в приложениях API'шках с burst нагрузкой (много запросов, а потом "засуха" → GC адаптируется)



# Практический эффект

- Выше пропускная способность
- Меньше задержки
- Можно заметить эффекты в приложениях API'шках с burst нагрузкой (много запросов, а потом "засуха" → GC адаптируется)
- Контейнеры лучше себя чувствуют → тратят меньше ресурсов



**Ставка идет на DATAS**  
Будущие улучшения  
мы увидим уже в нем





# JIT и его связь с GC



# Что поменялось?



# Что поменялось?

- Better escape analysis



# Что поменялось?

- Better escape analysis
- Deabstraction



# Что поменялось?

- Better escape analysis
- Deabstraction
- Devirtualisation



# Что поменялось?

- Better escape analysis
- Deabstraction
- Devirtualisation
- Better LINQ



# Что поменялось?

- Better escape analysis
- Deabstraction
- Devirtualisation
- Better LINQ
- Return Buffers



# Что поменялось?

- Better escape analysis
- Deabstraction
- Devirtualisation
- Better LINQ
- Return Buffers
- Над всем этим работают не первый год  
(про это отдельный доклад)





**Rule of thumb**  
Меньше аллокаций —  
проще жить GC



# Полезные ссылки

# Полезные ссылки

- [Статья](#) «Performance Improvements in .NET 10» от Стивена Тоуба
- В этой статье можно найти ссылки на интересные PR в dotnet runtime

# Полезные ссылки

- [Статья](#) «Performance Improvements in .NET 10» от Стивена Тоба
- В этой статье можно найти ссылки на интересные PR в dotnet runtime
- [Выступление](#) от Стивена Тоба на .NET Conf

# Полезные ссылки

- [Статья](#) «Performance Improvements in .NET 10» от Стивена Тоба
- В этой статье можно найти ссылки на интересные PR в dotnet runtime
- [Выступление](#) от Стивена Тоба на .NET Conf
- [Статья](#) о появлении DATAS
- [Статья](#) «Preparing for the .NET 10 GC (DATAS)» от Маони Стивенс

# Полезные ссылки

- [Статья](#) «Performance Improvements in .NET 10» от Стивена Тоба
- В этой статье можно найти ссылки на интересные PR в dotnet runtime
- [Выступление](#) от Стивена Тоба на .NET Conf
- [Статья](#) о появлении DATAS
- [Статья](#) «Preparing for the .NET 10 GC (DATAS)» от Маони Стивенс
- [Статья](#) «.NET 10 GC Write Barrier Improvements»

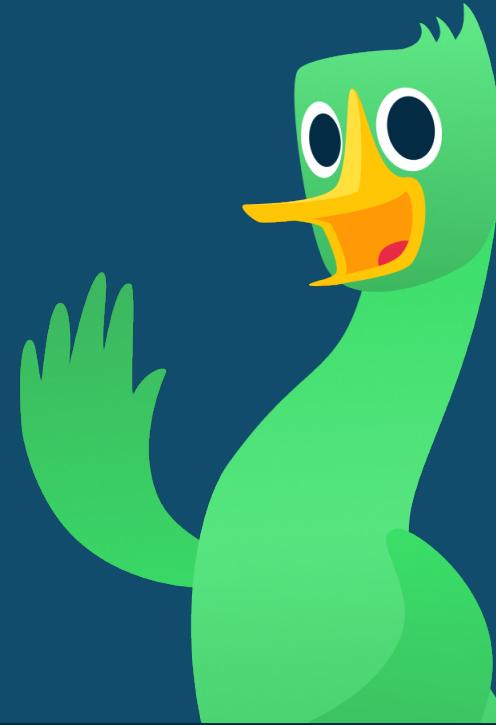


## Takeaways

Все делается  
«под капотом»,  
но улучшения есть 



Всем спасибо  
за внимание!



ozon{tech}



Антон Воронцов  
@williams750

