Яндекс

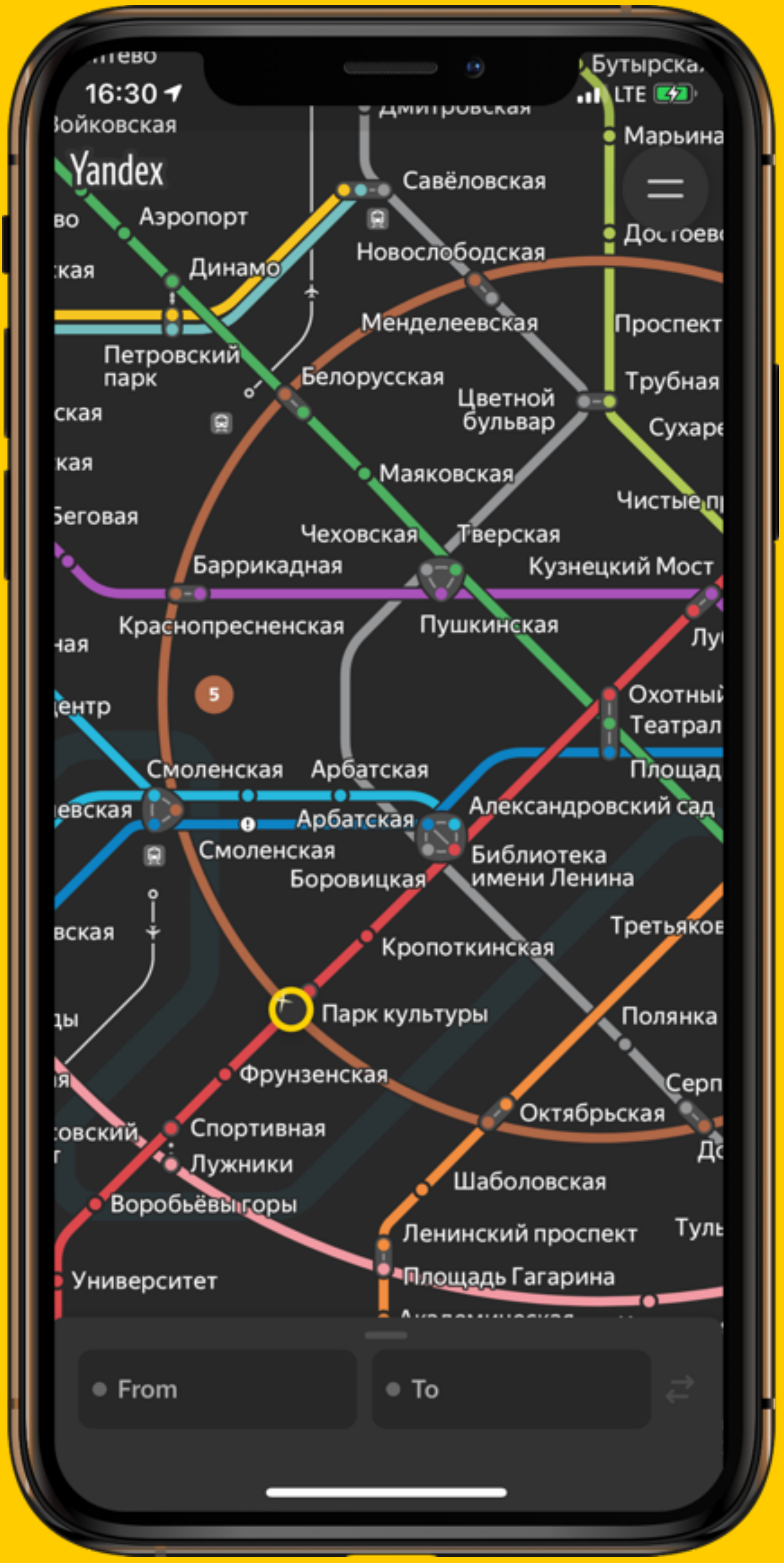
Яндекс

Scroll Mechanics

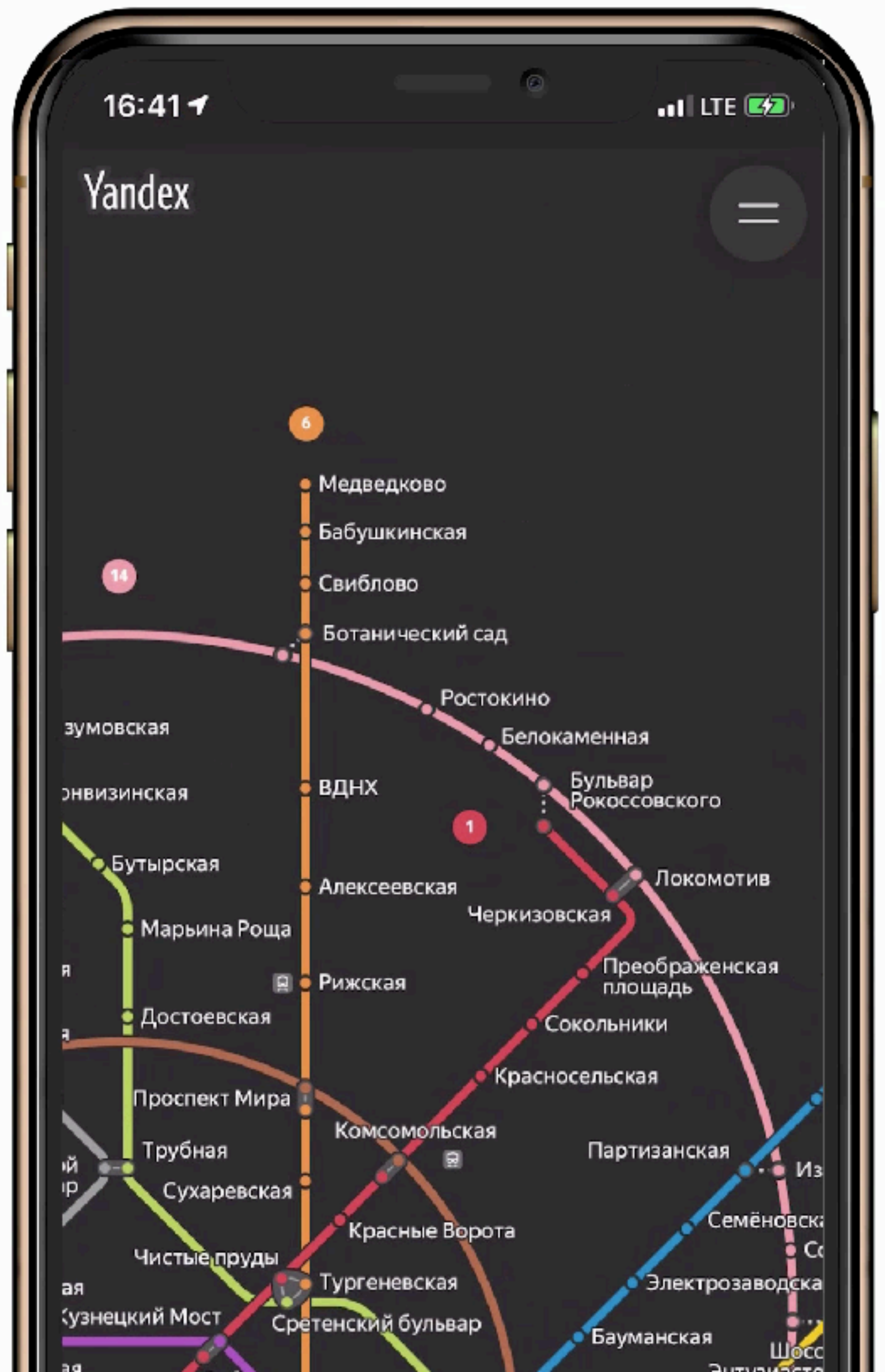
Илья Лобанов

Схема в Яндекс.Метро

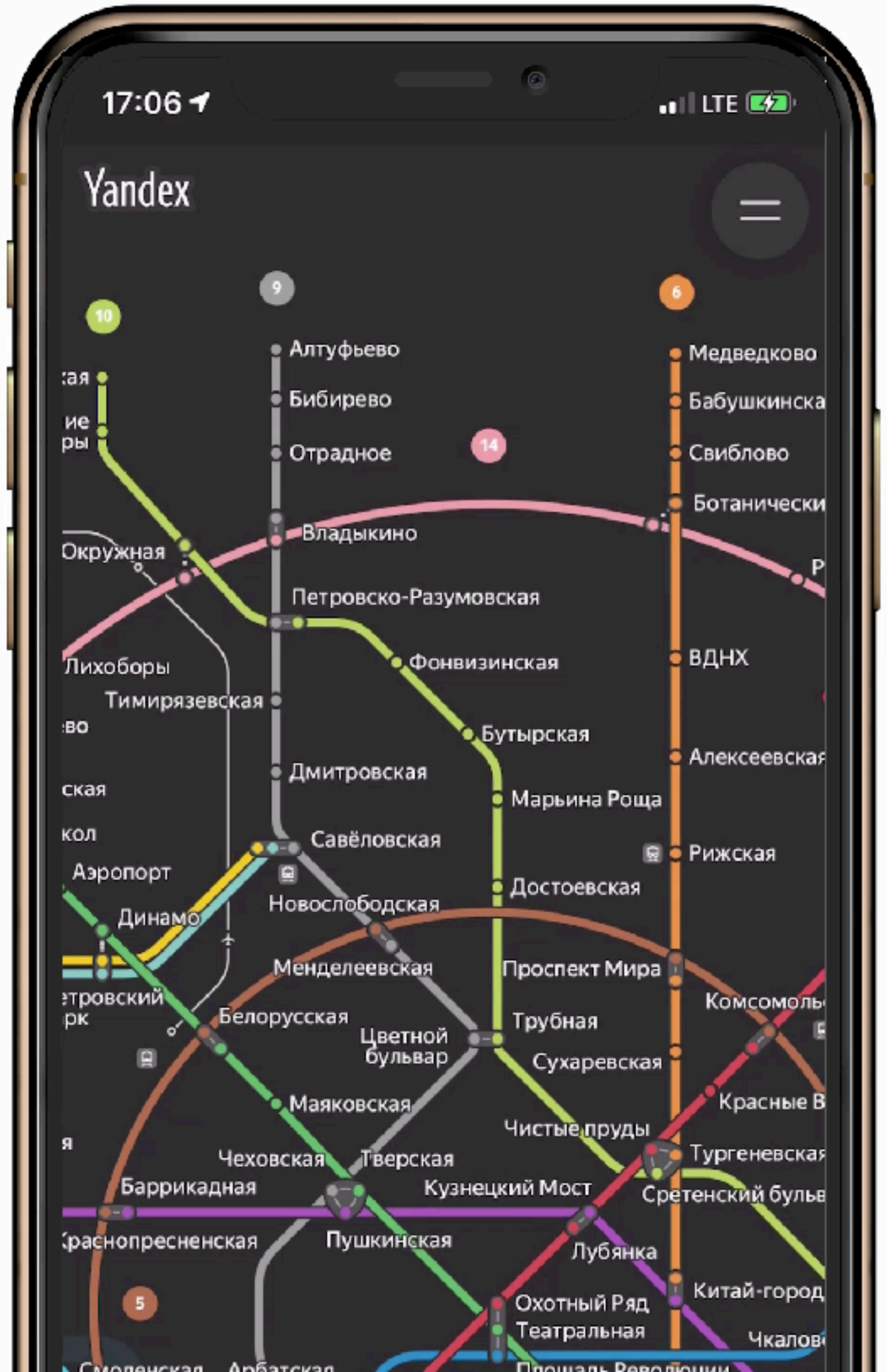
- Общая библиотека MetroKit для всех платформ
- Общее поведение скролла схемы
- В качестве референса был выбран UIScrollView



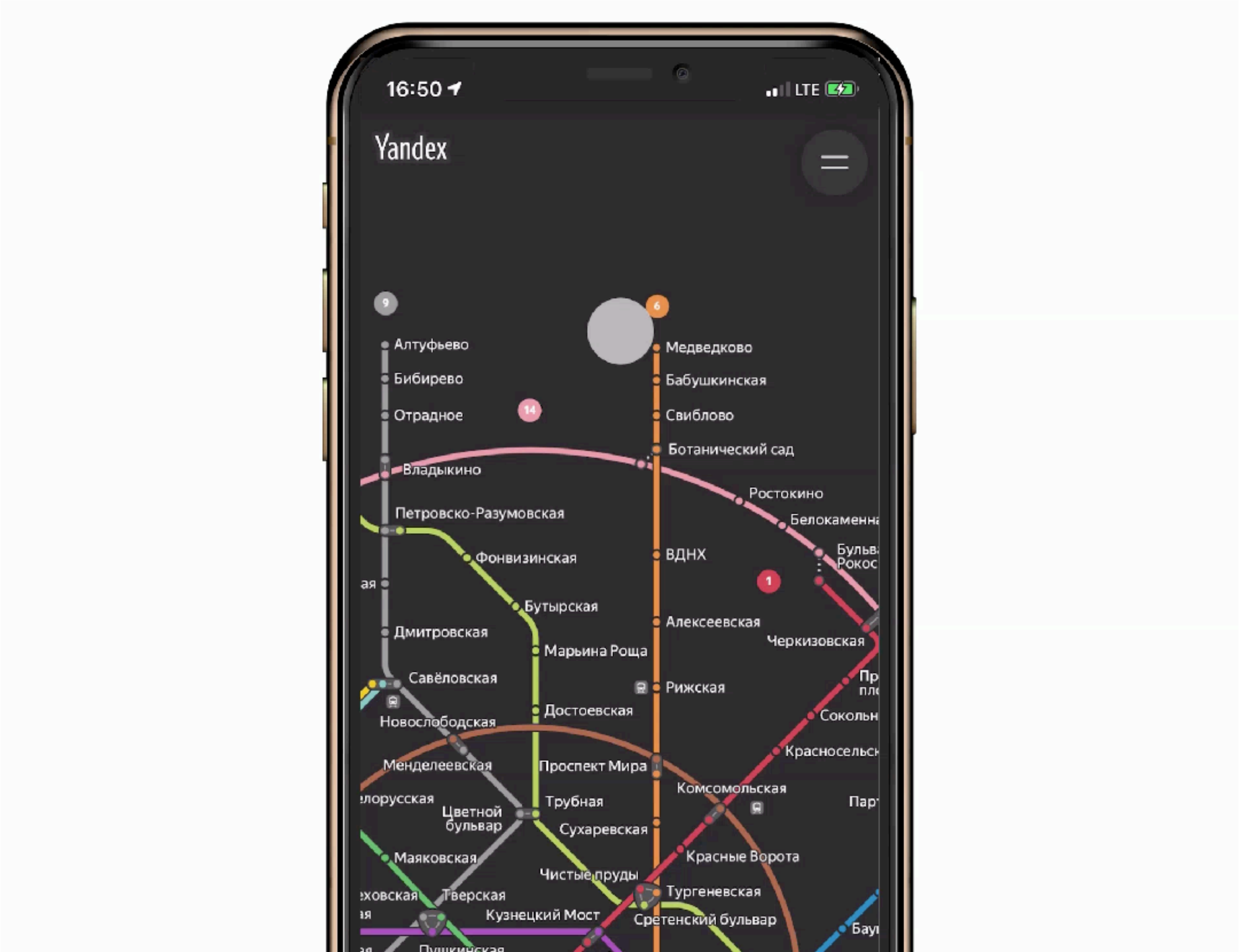
Механики UIScrollView Deceleration



Механики UIScrollView
Spring Animation



Механики UIScrollView
Rubber Band Effect



Содержание

01 | Тестовый пример

02 | Deceleration

03 | Spring Animation

04 | Rubber Band Effect

05 | Примеры

Тестовый пример

SimpleScrollView

```
class SimpleScrollView: UIView {
    var contentView: UIView?
    var contentSize: CGSize
    var contentOffset: CGPoint
}
```

SimpleScrollView

```
let panRecognizer = UIPanGestureRecognizer()

override init(frame: CGRect) {
    super.init(frame: frame)
    addGestureRecognizer(panRecognizer)
    panRecognizer.addTarget(self, action: #selector(handlePanRecognizer))
}
```

SimpleScrollView

```
enum State {
    case `default`
    case dragging(initialOffset: CGPoint)
}

var state: State = .default
```

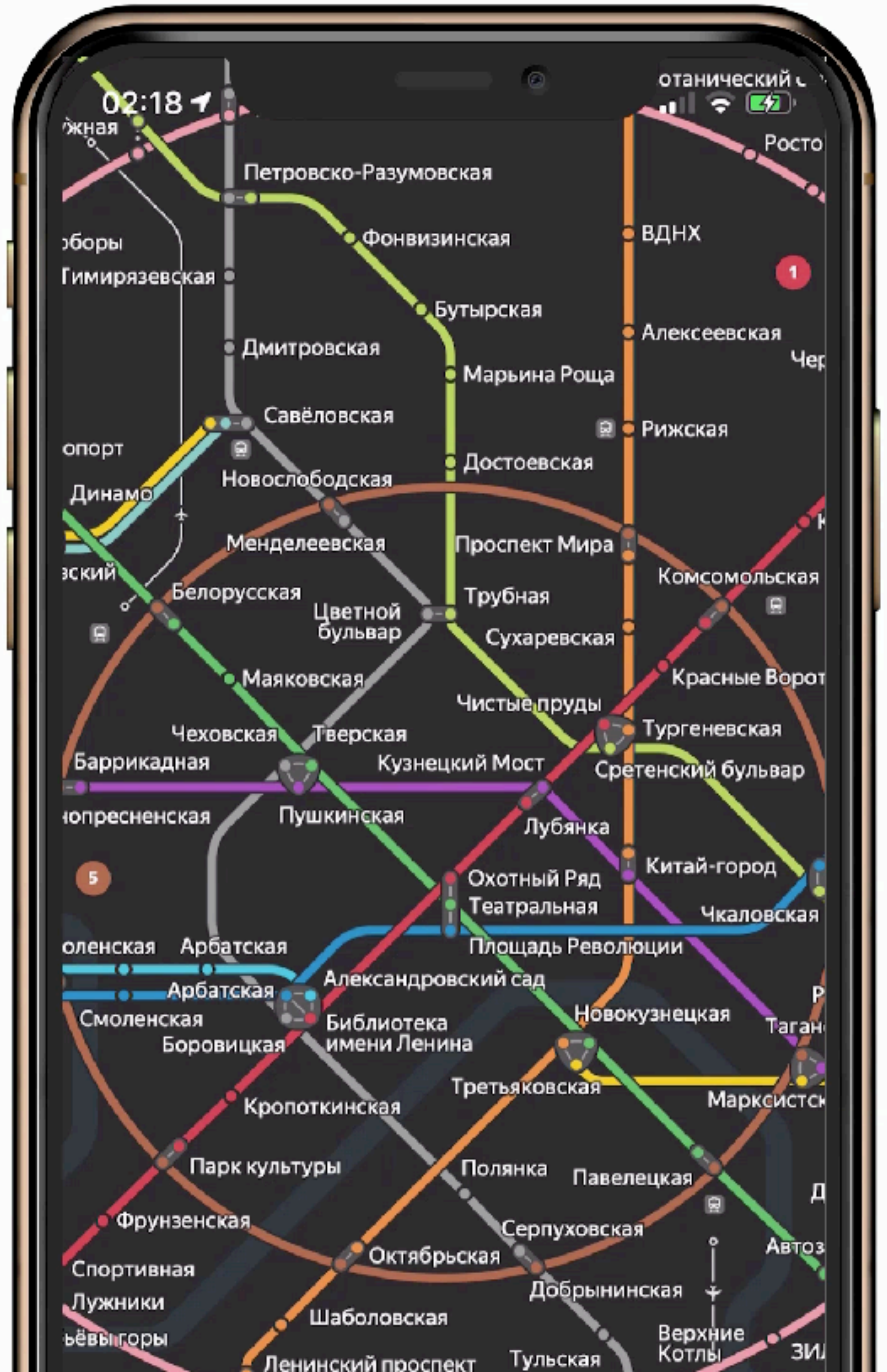
SimpleScrollView

```swift
@objc func handlePanRecognizer(_ sender: UIPanGestureRecognizer) {
    switch sender.state {
    case .began:
        state = .dragging(initialOffset: contentOffset)

    case .changed:
        let translation = sender.translation(in: self)
        if case .dragging(let initialOffset) = state {
            contentOffset = clampOffset(initialOffset - translation)
        }

    case .ended:
        state = .default

    // Other cases
    }
}
```

SimpleScrollView

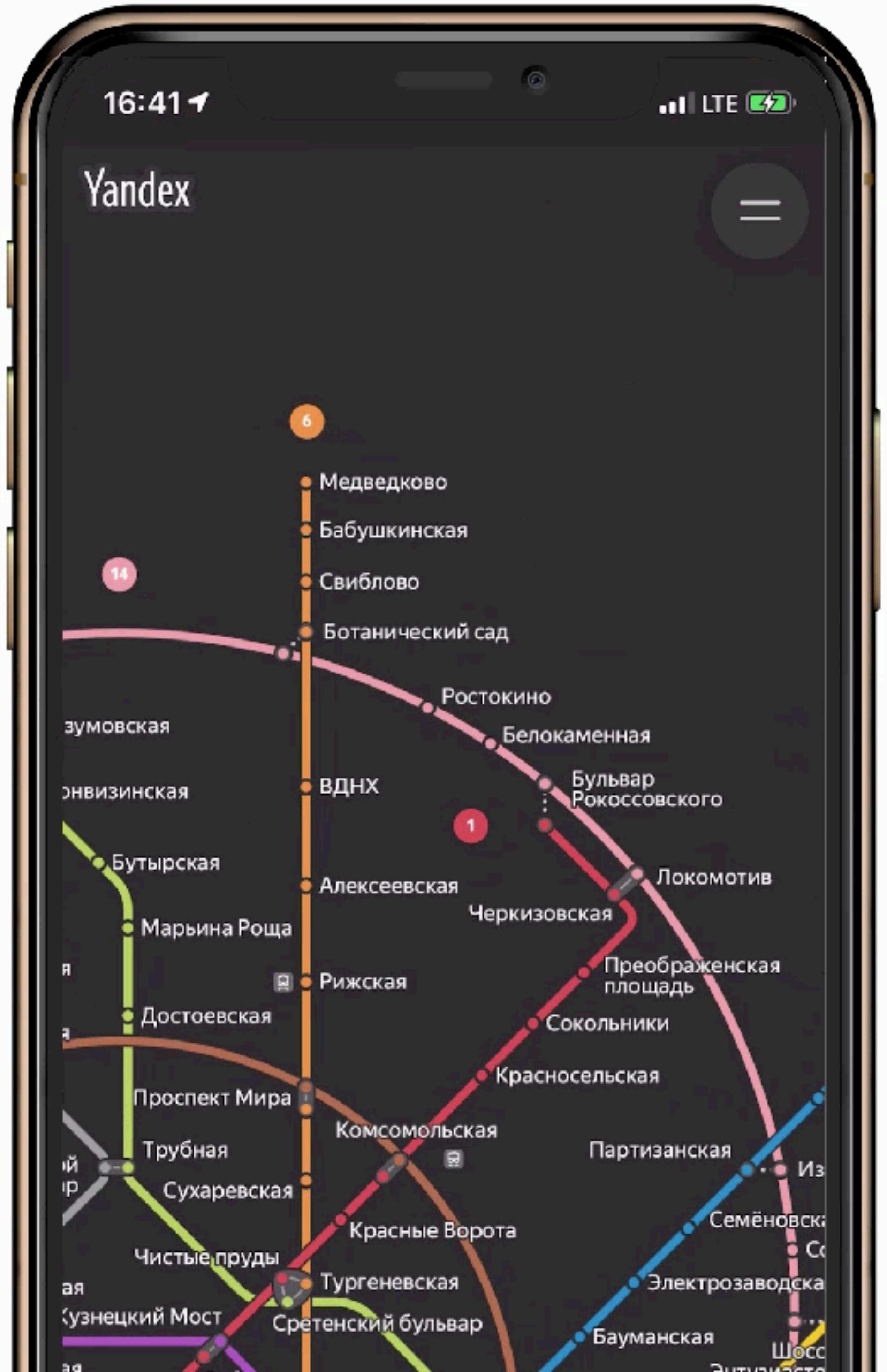
```
var contentOffsetBounds: CGRect {
    let width = contentSize.width - bounds.width
    let height = contentSize.height - bounds.height
    return CGRect(x: 0, y: 0, width: width, height: height)
}

func clampOffset(_ offset: CGPoint) -> CGPoint {
    return offset.clamped(to: contentOffsetBounds)
}
```

SimpleScrollView



Deceleration

Deceleration



Что известно про Deceleration

```
var decelerationRate: UIScrollView.DecelerationRate
```

> A floating-point value that determines the rate of deceleration after the user lifts their finger

```
extension UIScrollView.DecelerationRate {
    static let normal: UIScrollView.DecelerationRate // 0.998
    static let fast: UIScrollView.DecelerationRate // 0.99
}
```

UIScrollView | developer.apple.com/documentation/uikit/uiscrollview

Что известно про Deceleration

- Финальная позиция скролла
  (Designing Fluid Interfaces — WWDC 2018):

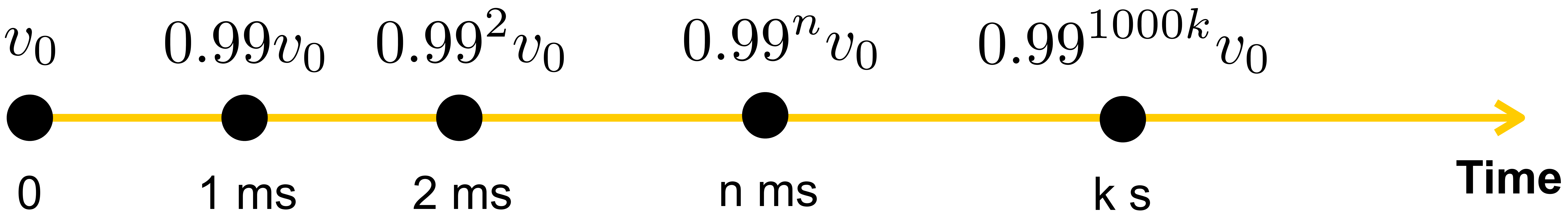
```swift
// Distance travelled after decelerating to zero velocity at a constant rate.
func project(initialVelocity: Float, decelerationRate: Float) -> Float {
    return (initialVelocity / 1000.0) * decelerationRate / (1.0 - decelerationRate)
}
```

Designing Fluid Interfaces | developer.apple.com/videos/play/wwdc2018/803

Изменение скорости

Deceleration Rate

```
.fast = 0.99
```



v_0 $0.99v_0$ 0.99^2v_0 0.99^nv_0 $0.99^{1000k}v_0$

0 1 ms 2 ms n ms k s

Time

Функция скорости

$$v(t) = v_0 d^{1000t}$$

- v — скорость, pt/s
- v_0 — начальная скорость, pt/s
- d — коэффициент замедления (*0* < *d* < *1*)
- t — время, s

Уравнение движения

$x(t) =?$

$$x(t) = x_0 + \int_0^t v(x)dx$$

x_0 — начальное положение точки

Уравнение движения

$$x(t) = x_0 + v_0 \int_0^t d^{1000x} dx$$

$$x(t) = x_0 + v_0 \frac{d^{1000t} - 1}{1000 ln(d)}$$

Конечная точка

$$X = \lim_{t\to\infty} x(t) = \lim_{t\to\infty} (x_0 + v_0 \frac{d^{1000t} - 1}{1000ln(d)})$$

$$X = x_0 - \frac{v_0}{1000ln(d)}$$

Конечная точка

$$X = x_0 - \frac{v_0}{1000ln(d)}$$

Apple:

$$X = x_0 + \frac{v_0 d}{1000(1 - d)}$$

Сравнение с формулой от Apple

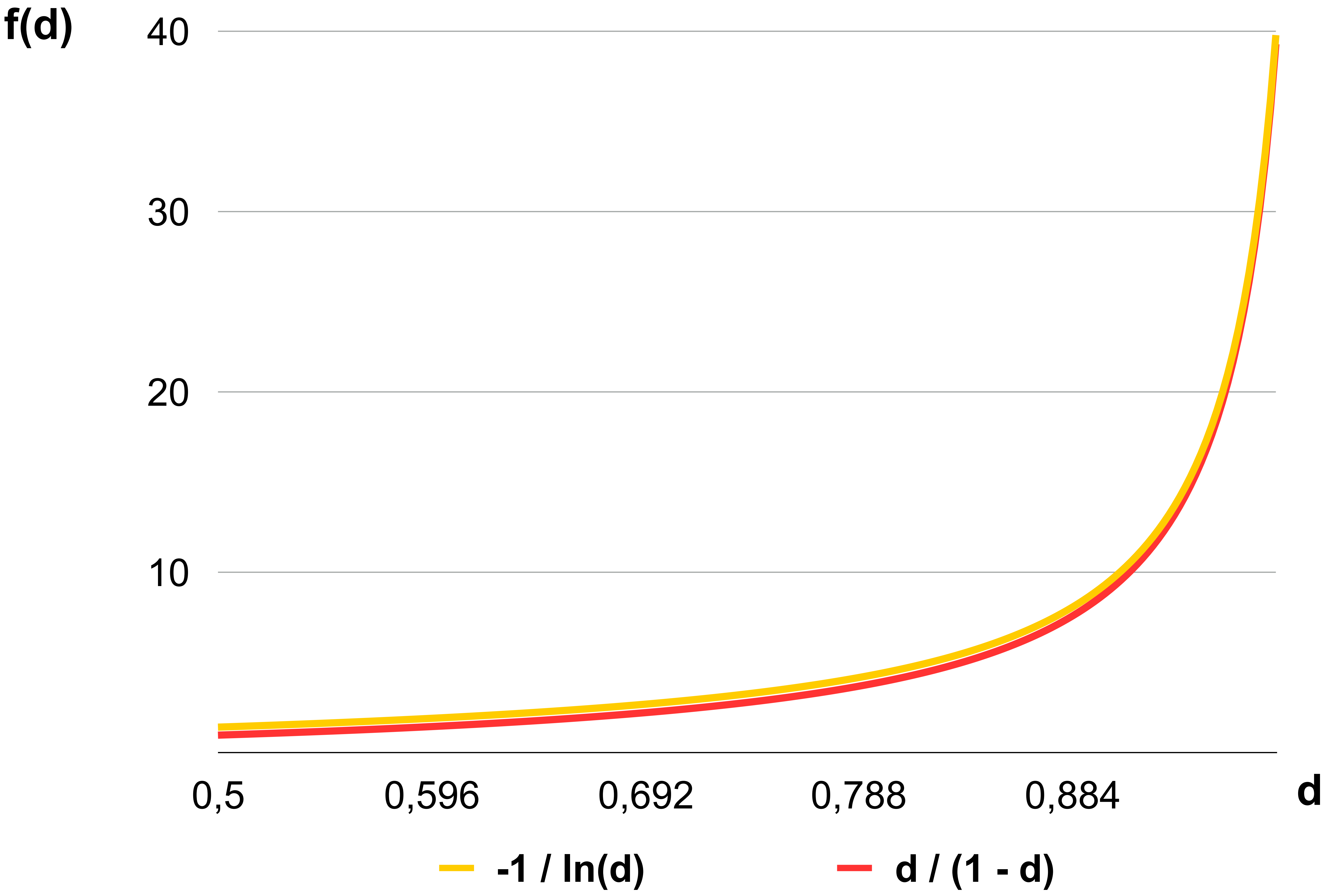
$$\frac{-1}{ln(d)} \quad \text{vs} \quad \frac{d}{1-d}$$

- Разложение натурального логарифма в ряд Тейлора:

$$ln(x) = \frac{x-1}{x} + \frac{(x-1)^2}{2x^2} + \ldots$$

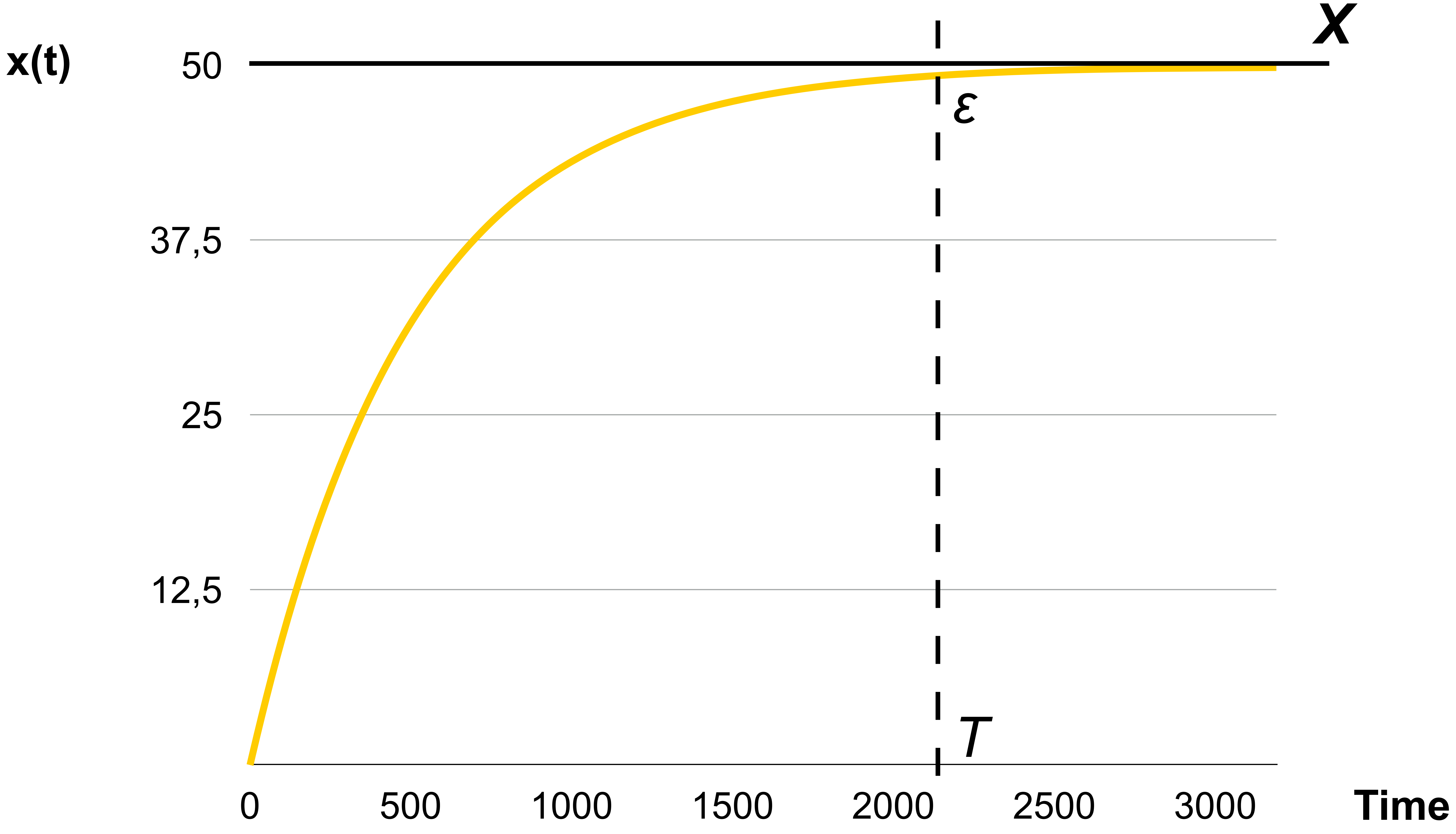
Natural logarithm | en.wikipedia.org/wiki/Natural_logarithm#Series

Сравнение с формулой от Apple



```
.normal = 0.998
.fast = 0.99
```

Время движения



Время движения

$$|X - x(T)| = \varepsilon$$

$$T = \frac{ln(\frac{-1000\varepsilon ln(d)}{|v_0|})}{1000ln(d)}$$

DecelerationTimingParameters

```
struct DecelerationTimingParameters {
    var initialValue: CGPoint
    var initialVelocity: CGPoint
    var decelerationRate: CGFloat
    var threshold: CGFloat
}


extension DecelerationTimingParameters {
    var destination: CGPoint
    var duration: TimeInterval
    func value(at time: TimeInterval) -> CGPoint
}
```

TimerAnimation

```swift
class TimerAnimation {
    typealias Animations = (_ progress: Double, _ time: TimeInterval) -> Void
    typealias Completion = (_ finished: Bool) -> Void

    init(duration: TimeInterval, animations: @escaping Animations,
         completion: Completion? = nil)
}
```

SimpleScrollView

```swift
@objc func handlePanRecognizer(_ sender: UIPanGestureRecognizer) {
    switch sender.state {
    case .began:
        state = .dragging(initialOffset: contentOffset)

    case .changed:
        let translation = sender.translation(in: self)
        if case .dragging(let initialOffset) = state {
            contentOffset = clampOffset(initialOffset - translation)
        }

    case .ended:
        state = .default



    // Other cases
    }
}
```

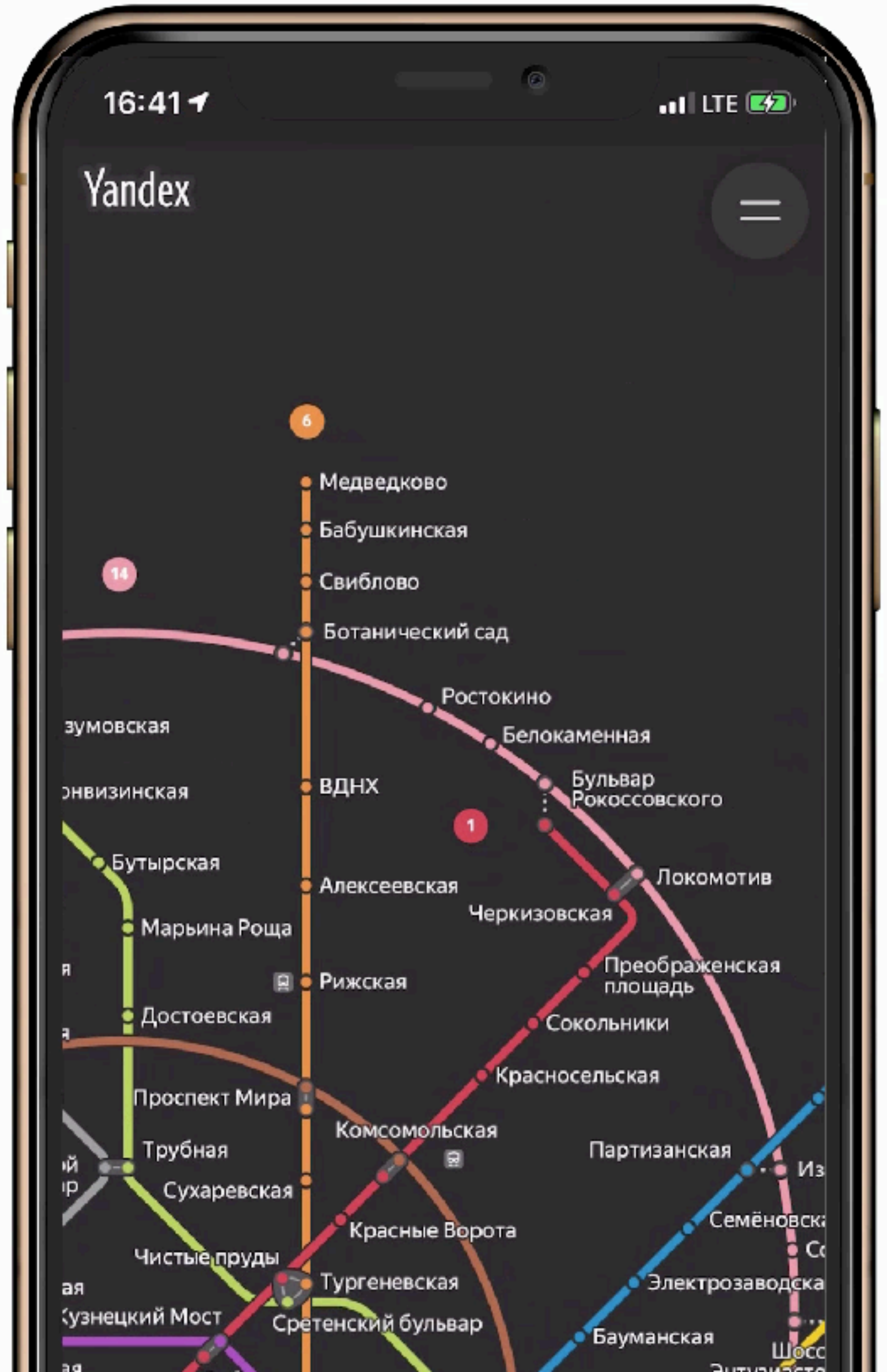
SimpleScrollView

```
@objc func handlePanRecognizer(_ sender: UIPanGestureRecognizer) {
    switch sender.state {
    case .began:
        state = .dragging(initialOffset: contentOffset)

    case .changed:
        let translation = sender.translation(in: self)
        if case .dragging(let initialOffset) = state {
            contentOffset = clampOffset(initialOffset - translation)
        }

    case .ended:
        state = .default
        let velocity = sender.velocity(in: self)
        startDeceleration(withVelocity: -velocity)

    // Other cases
    }
}
```

SimpleScrollView

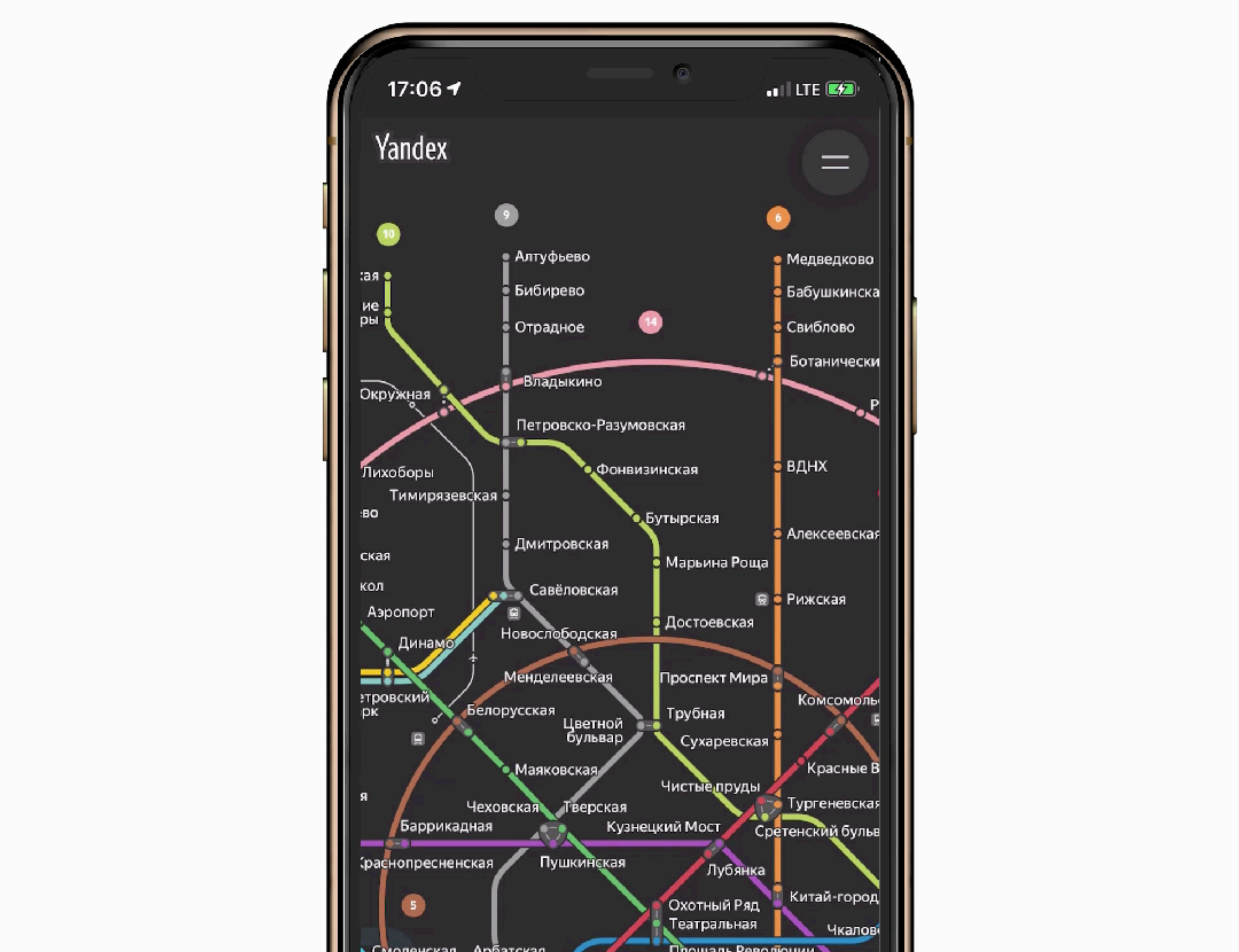
```
var contentOffsetAnimation: TimerAnimation?

func startDeceleration(withVelocity velocity: CGPoint) {
    let decelerationRate = UIScrollView.DecelerationRate.normal.rawValue
    let threshold = 0.5 / UIScreen.main.scale

    let parameters = DecelerationTimingParameters(initialValue: contentOffset,
                                                  initialVelocity: velocity,
                                                  decelerationRate: decelerationRate,
                                                  threshold: threshold)


    contentOffsetAnimation = TimerAnimation(
        duration: parameters.duration,
        animations: { [weak self] _, time in
            guard let self = self else { return }
            self.contentOffset = self.clampOffset(parameters.value(at: time))
        })
}
```

Deceleration



Spring Animation

Spring Animation

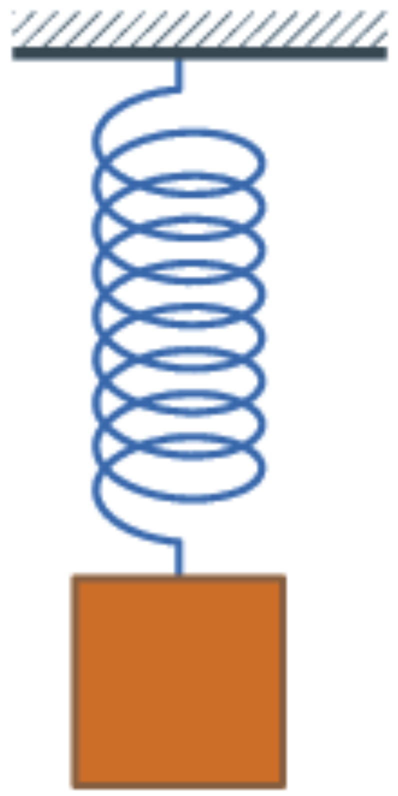


Spring Animation

- m — масса (mass)
- k — жесткость (stiffness)
- d — затухание (damping)

  или
- ζ — коэффициент затухания (damping ratio)

$$\zeta = \frac{d}{2\sqrt{km}}$$



Damping ratio | en.wikipedia.org/wiki/Damping_ratio

Уравнение движения

$$m\frac{d^2x}{dt^2} + d\frac{dx}{dt} + kx = 0$$

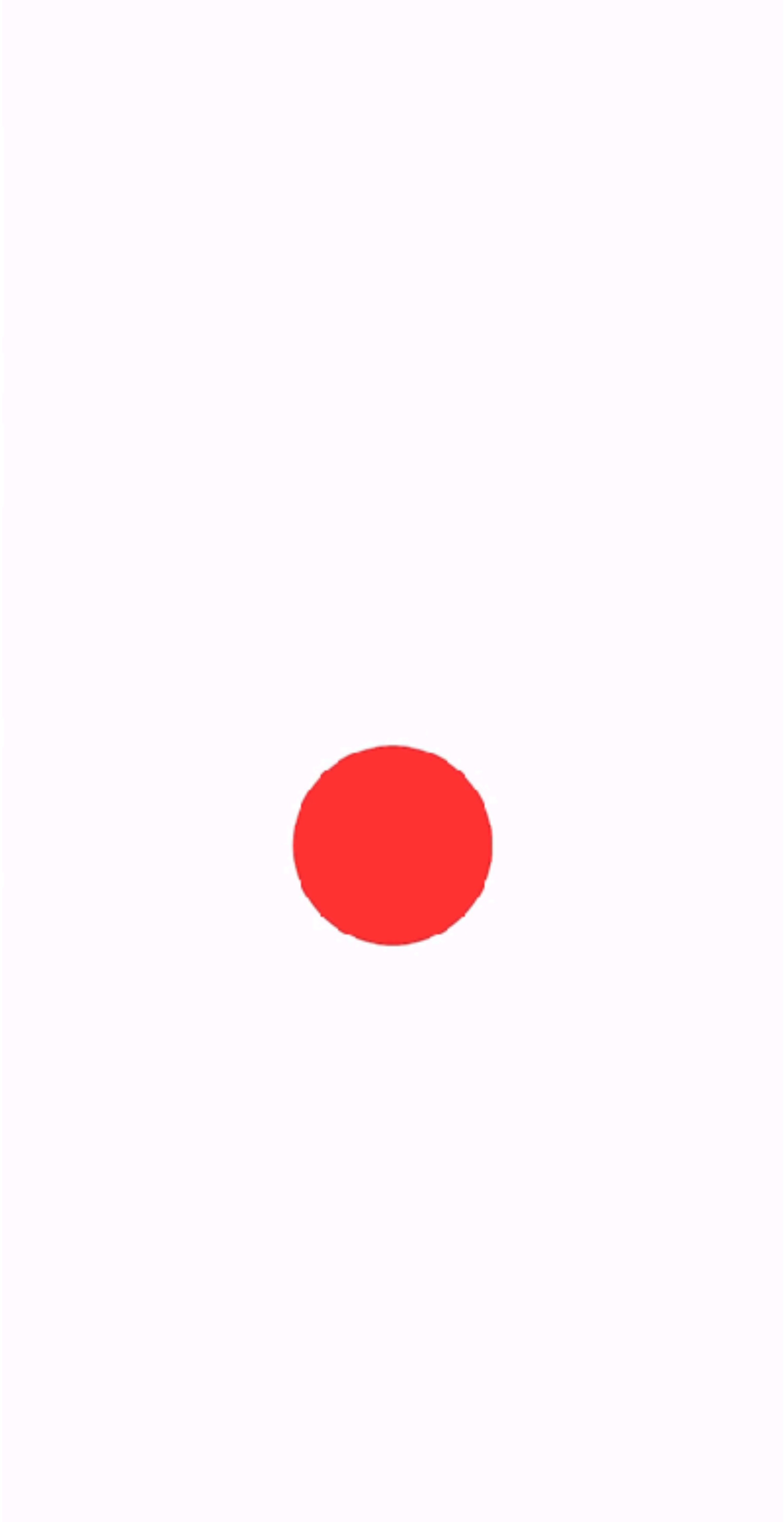
- m — масса (mass)
- k — жесткость (stiffness)
- d — затухание (damping)

Damping ratio | en.wikipedia.org/wiki/Damping_ratio

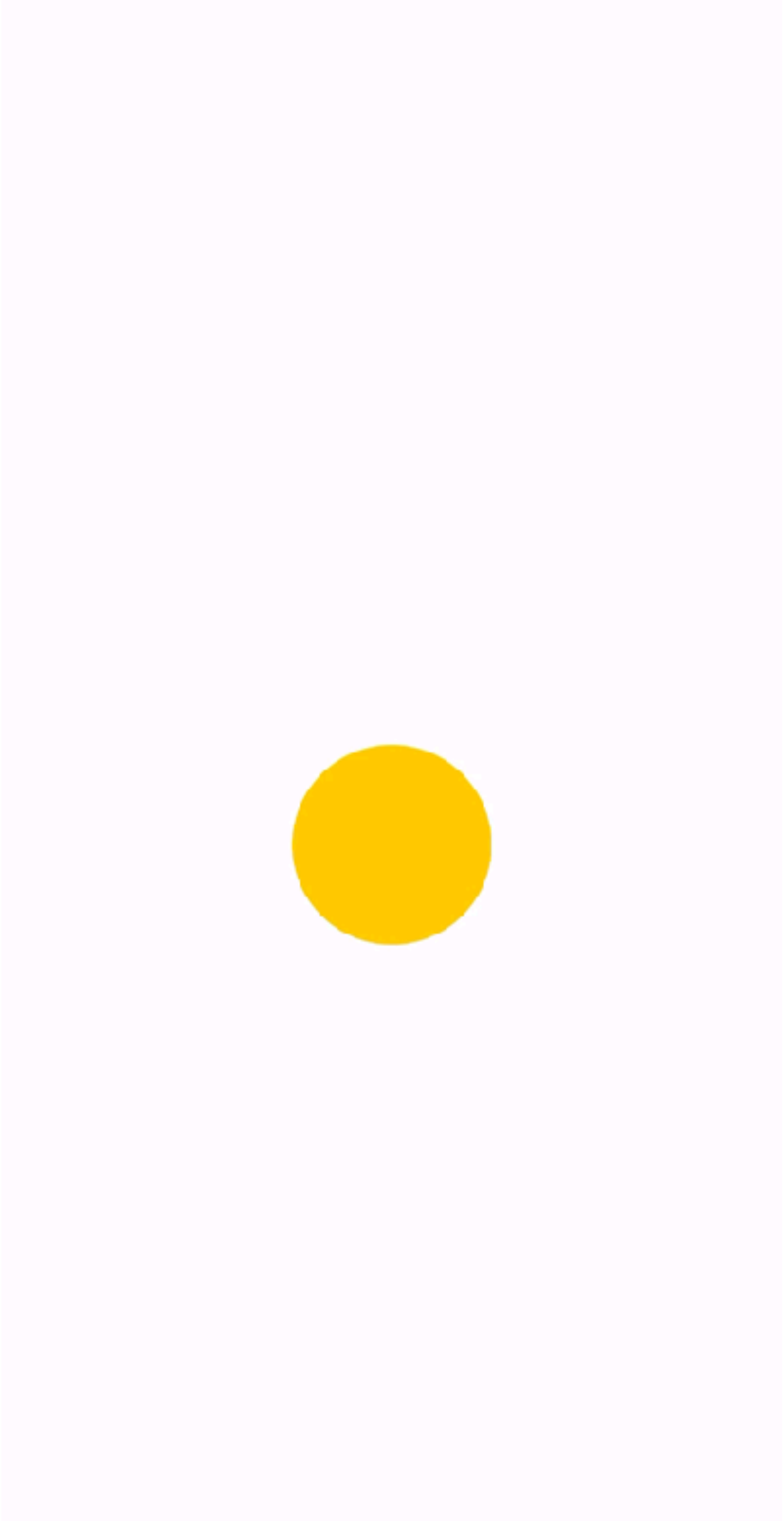
Damping Ratio ζ

0.1

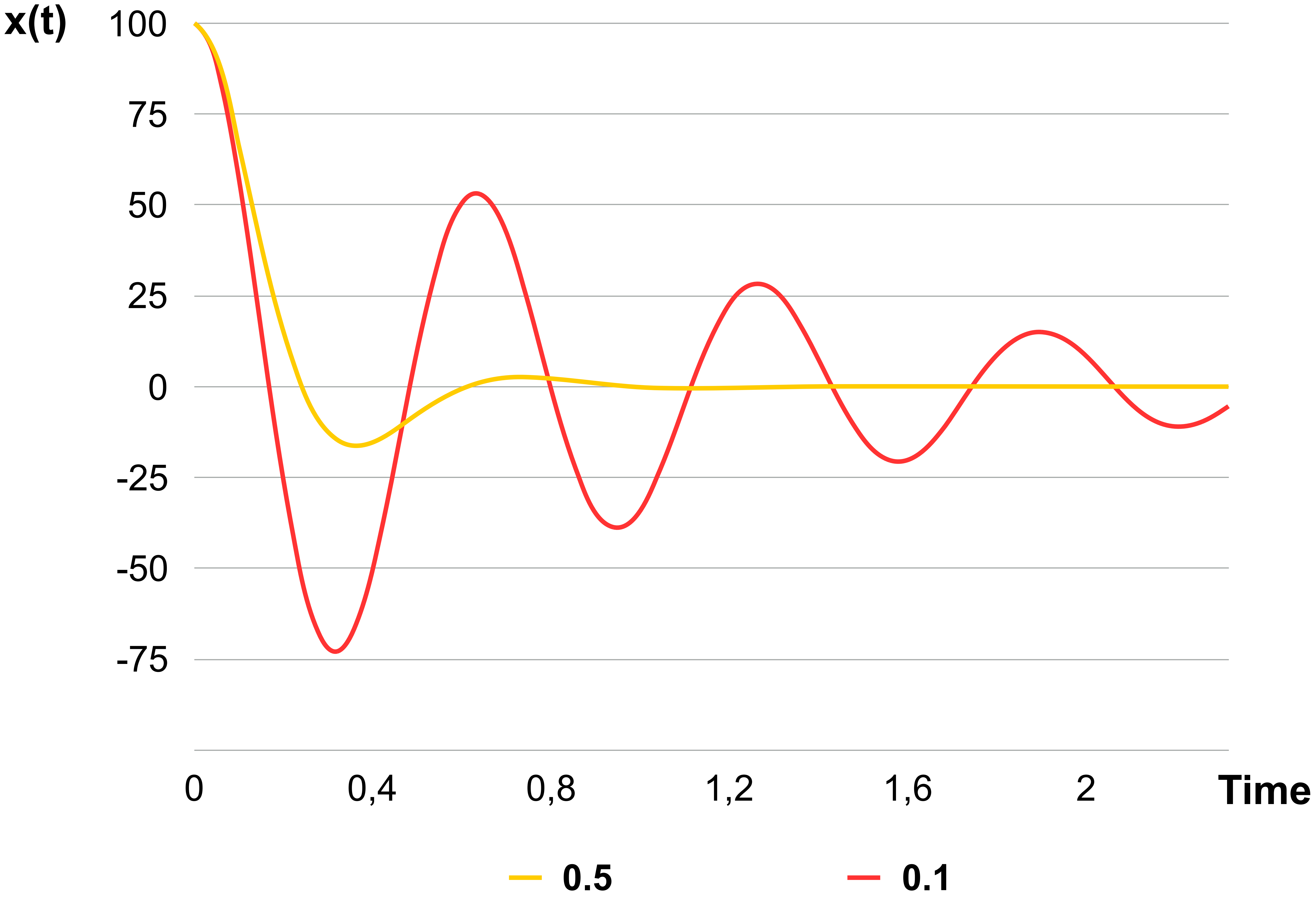
0.5



1.0

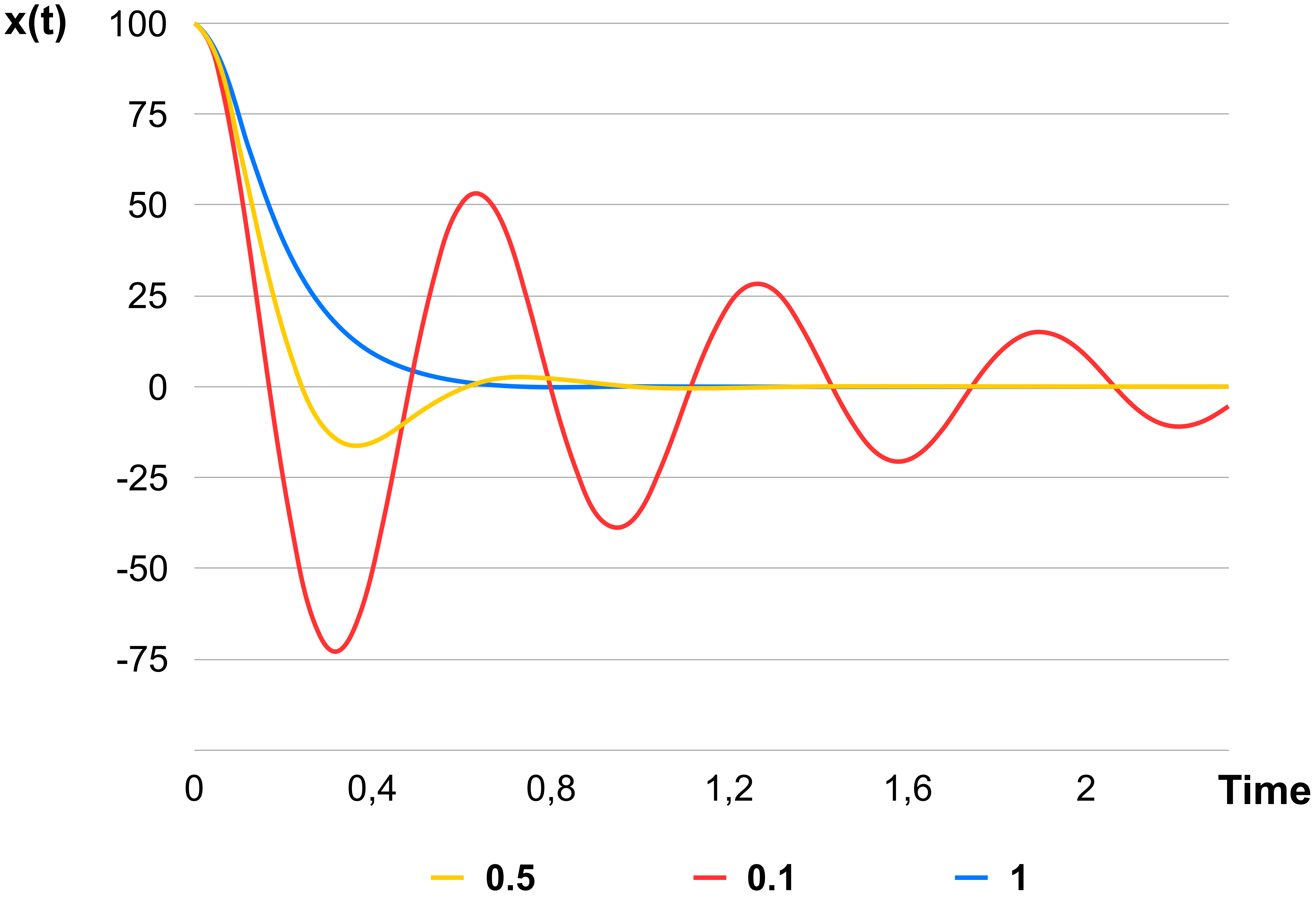


Damping Ratio ζ



- Слабое затухание (Underdamped)
  $0 < \zeta < 1$

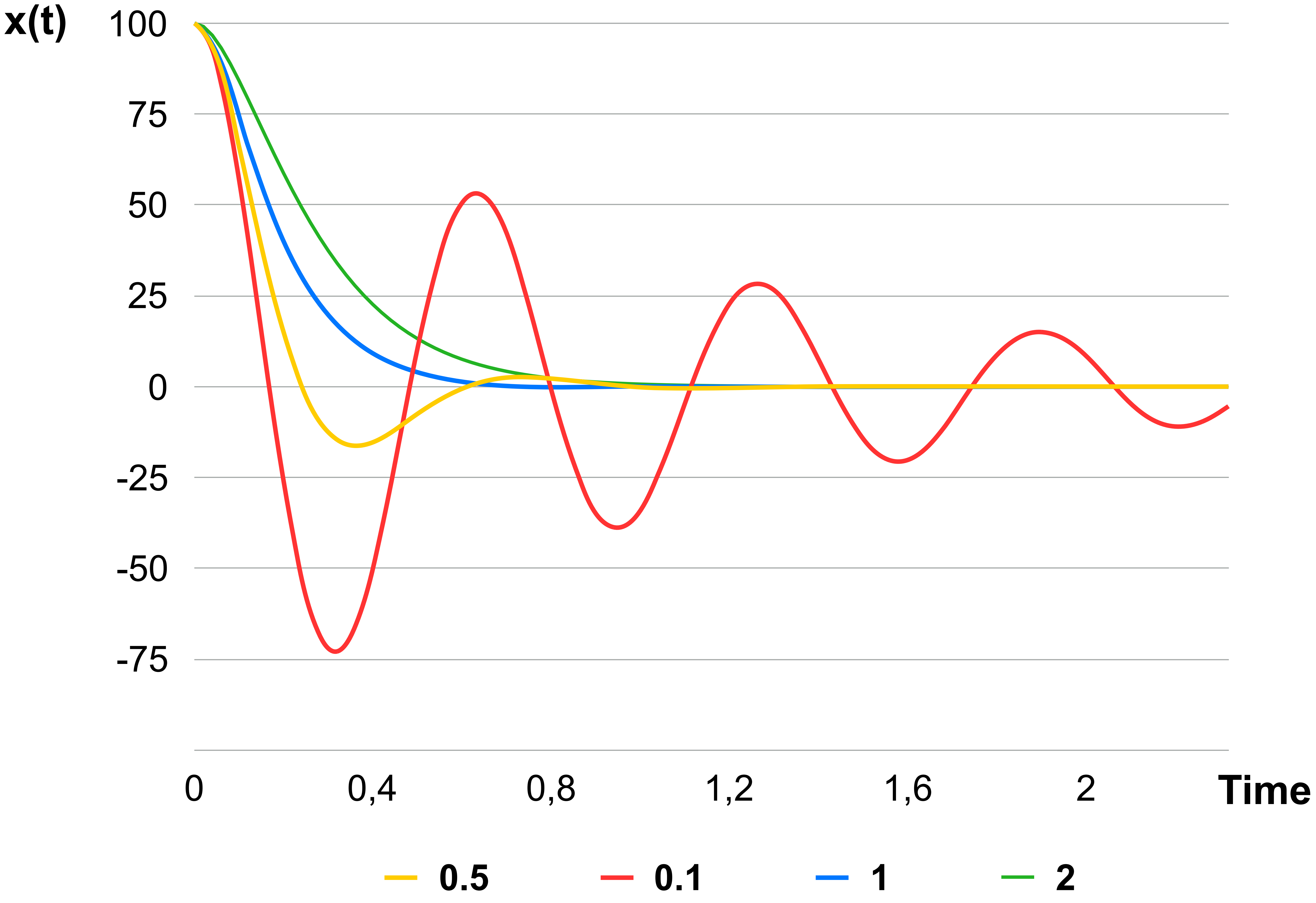
Damping Ratio ζ



- Слабое затухание (Underdamped) ***$0 < \zeta < 1$***

- Граница апериодичности (Critically damped) ***$\zeta = 1$***

Damping Ratio ζ



- Слабое затухание (Underdamped)
  0 < ζ < 1
- Граница апериодичности (Critically damped)
  ζ = 1
- Апериодичность (Overdamped)
  ζ > 1

Уравнение движения

$$m\frac{d^2x}{dt^2} + d\frac{dx}{dt} + kx = 0$$

- Уравнение движения

$$x(t) = ?$$

- Время колебаний

$$T = ?$$

Уравнение движения
Слабое затухание (0 < ζ < 1)

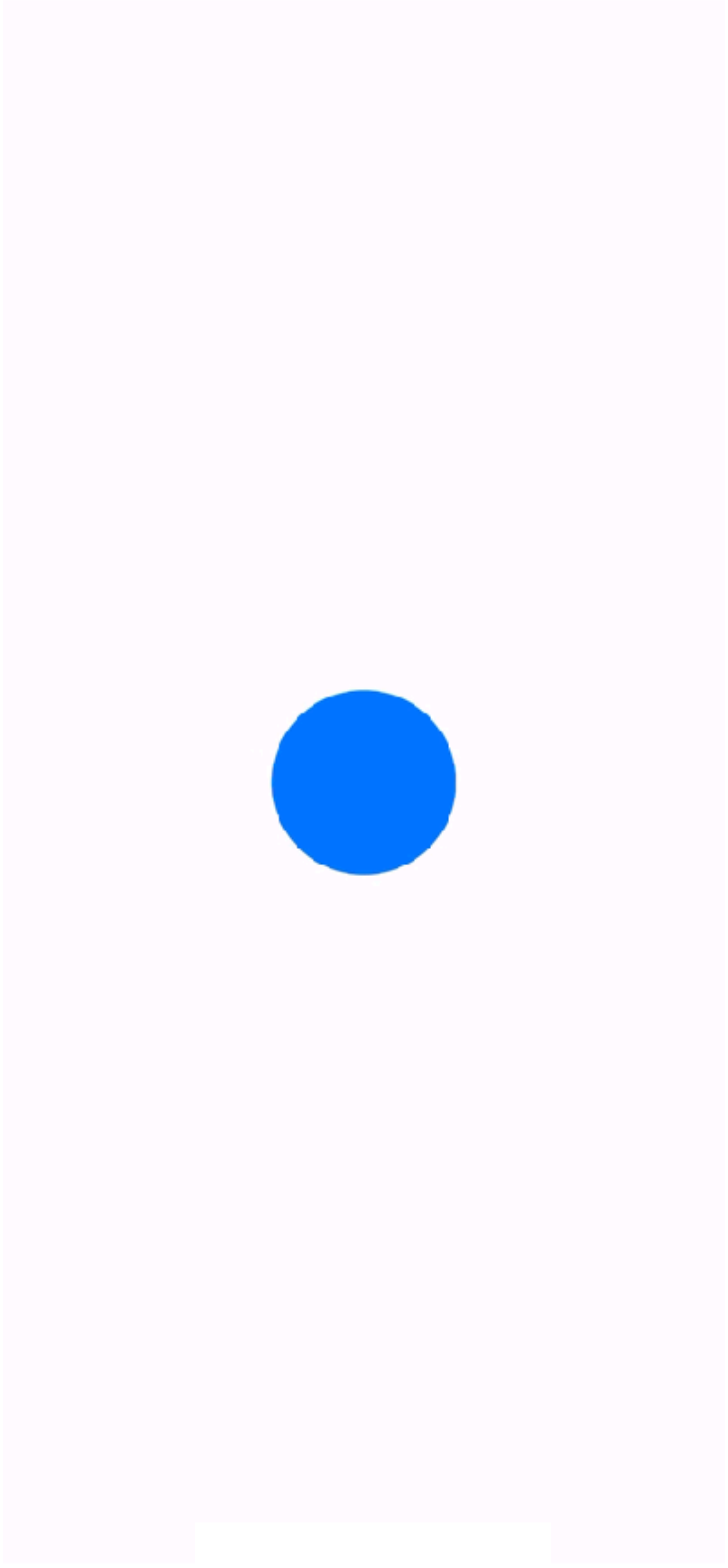
$$x(t) = (C_1 cos\omega' t + C_2 sin\omega' t)e^{-\beta t}$$

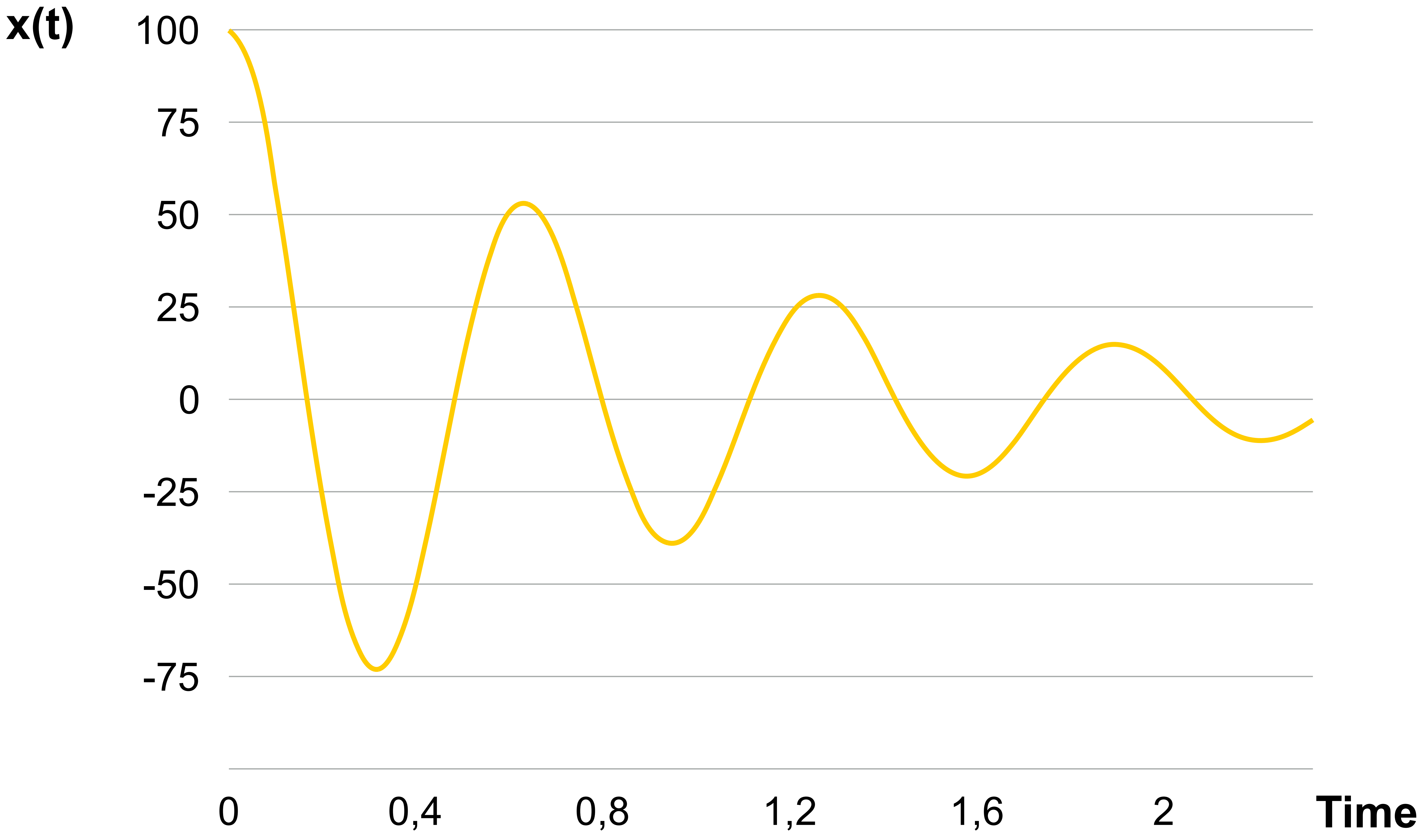
ω' — собственная частота системы (damped natural frequency)

$$\beta = \frac{d}{2m}$$

$$\begin{aligned} x(0) &= x_0 \\ x'(0) &= v_0 \end{aligned} \Rightarrow \begin{aligned} C_1 &= x_0 \\ C_2 &= \frac{v_0 + \beta x_0}{\omega'} \end{aligned}$$

Уравнение движения
Слабое затухание (0 < ζ < 1)





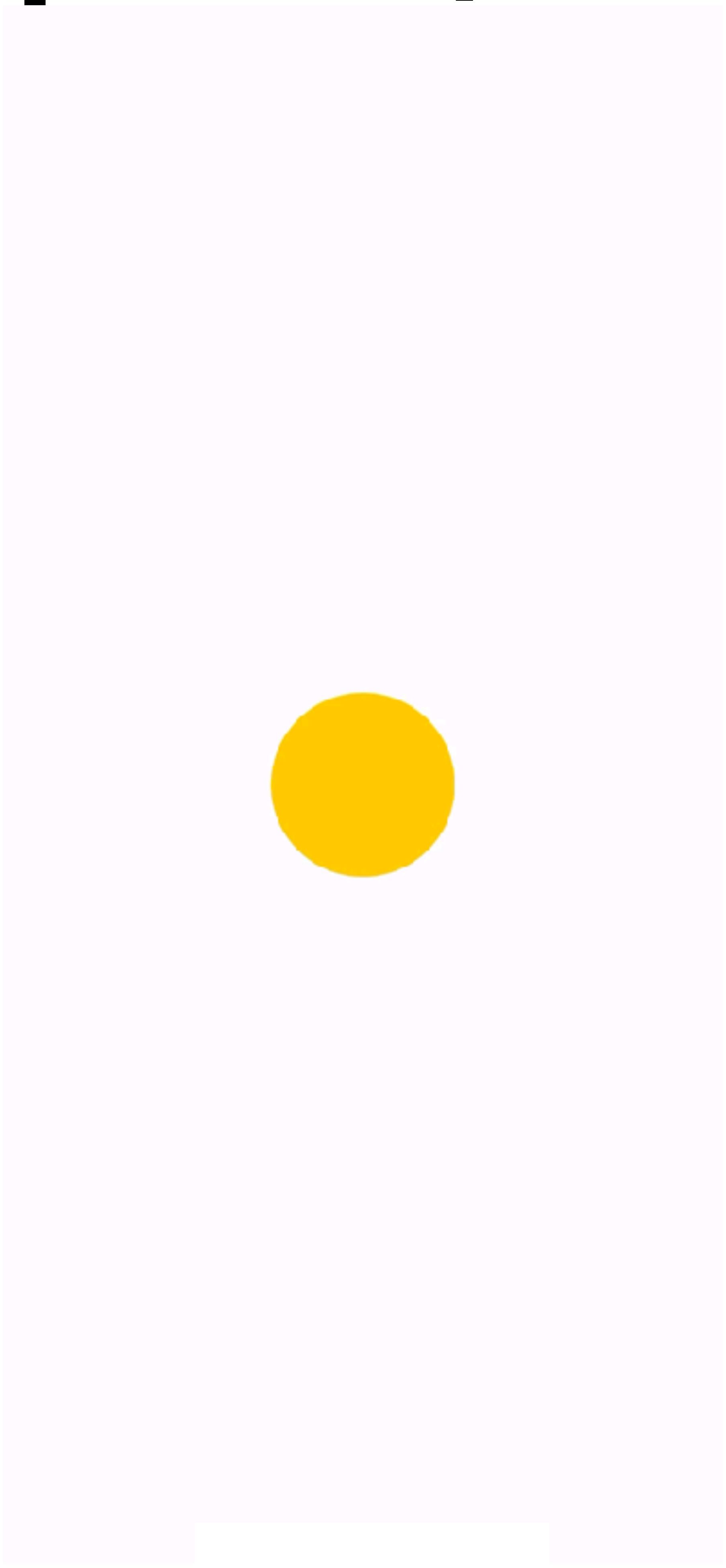
Уравнение движения
Граница апериодичности (ζ = 1)

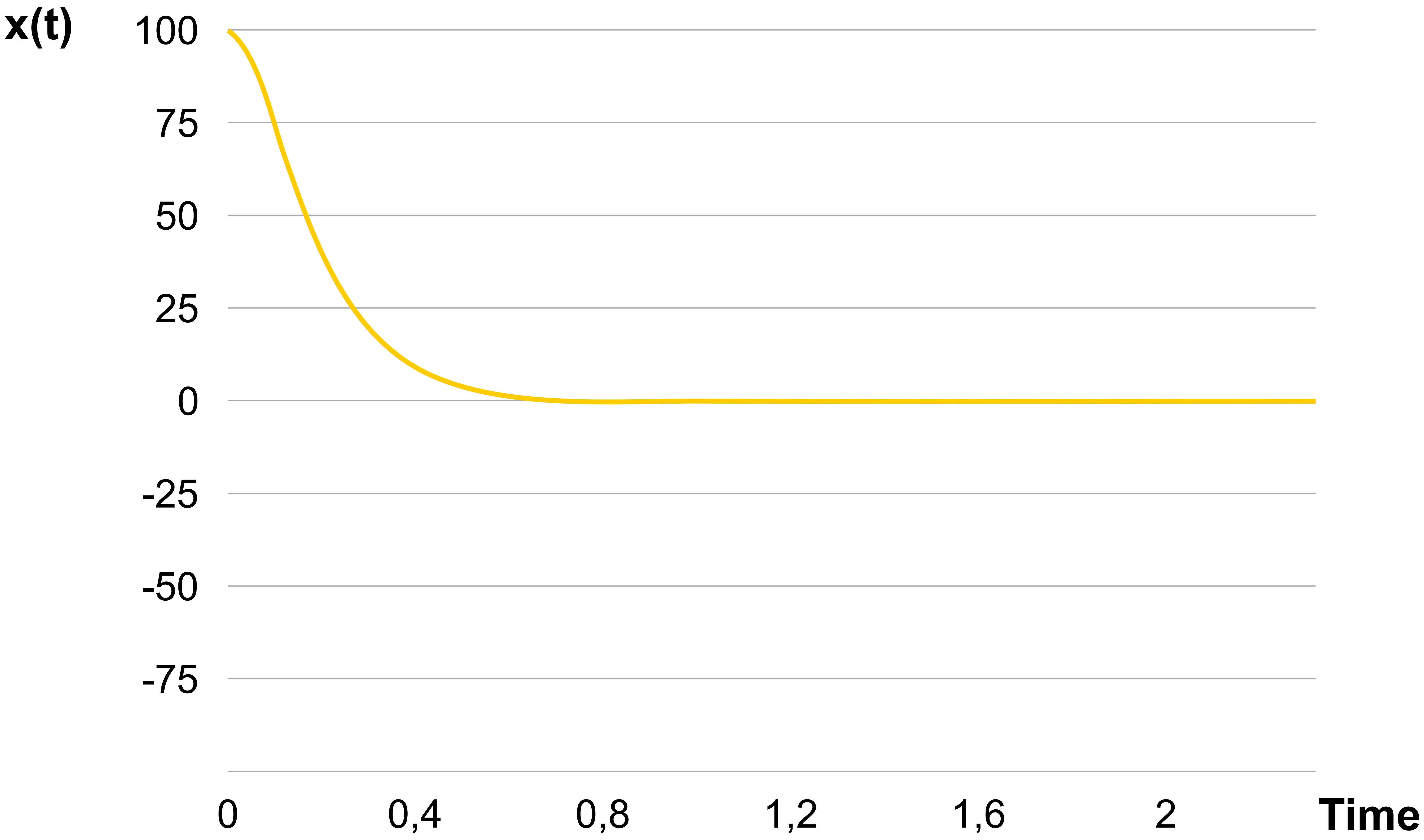
$$x(t) = (C_1 + C_2 t)e^{-\beta t}$$

$$\beta = \frac{d}{2m}$$

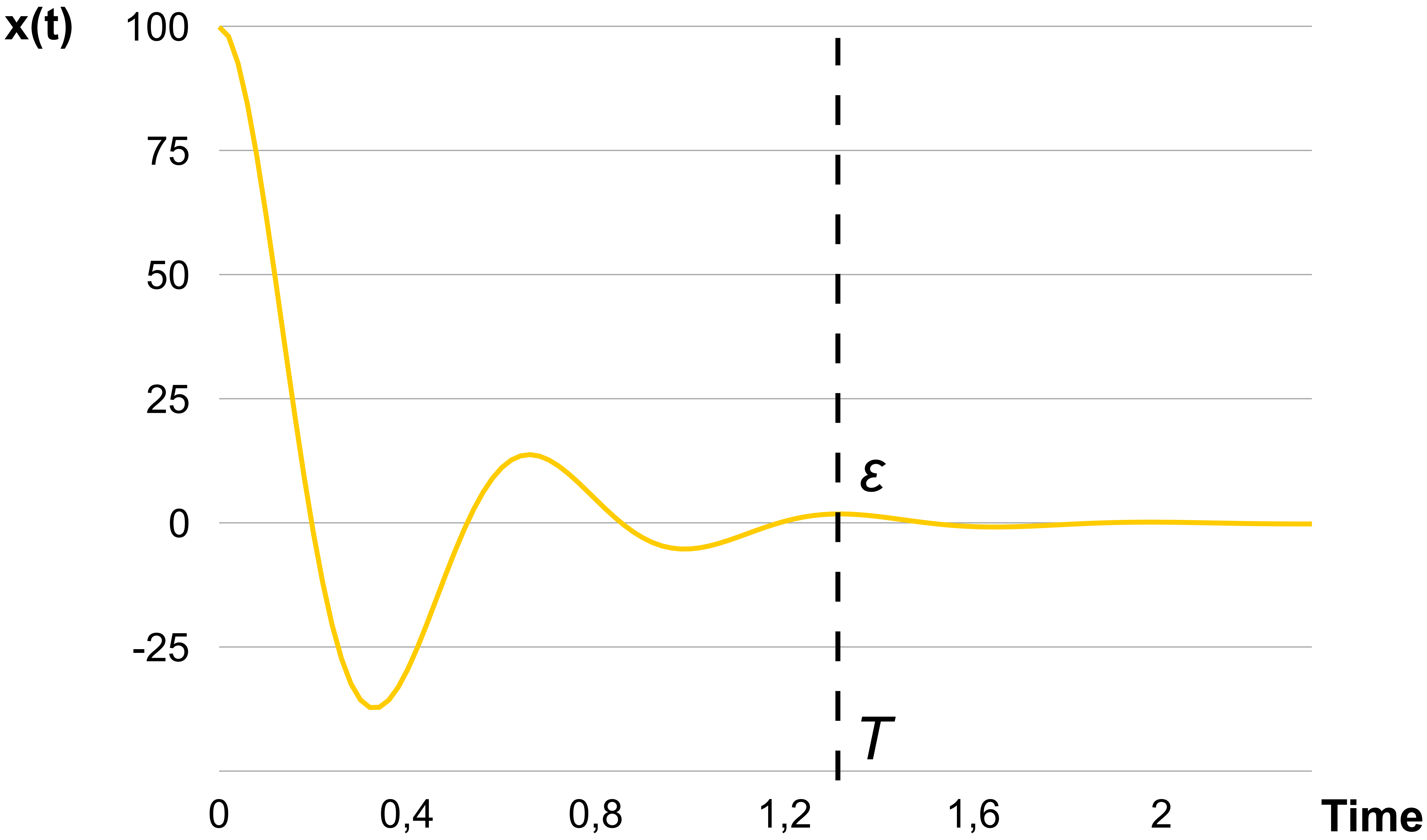
$$\begin{matrix} x(0) = x_0 \\ x'(0) = v_0 \end{matrix} \Rightarrow \begin{matrix} C_1 = x_0 \\ C_2 = v_0 + \beta x_0 \end{matrix}$$

Уравнение движения
Граница апериодичности (ζ = 1)





Время колебаний



Время колебаний
Слабое затухание (0 < ζ < 1)

$$T = \frac{1}{\beta} ln \frac{|C_1| + |C_2|}{\varepsilon}$$

$$\beta = \frac{d}{2m}$$

$$C_1 = x_0$$

$$C_2 = \frac{v_0 + \beta x_0}{\omega'}$$

Время колебаний
Граница апериодичности (ζ = 1)

$$T = max(\frac{1}{\beta} ln \frac{2|C_1|}{\varepsilon}, \frac{2}{\beta} ln \frac{4|C_2|}{e\beta\varepsilon})$$

$$\beta = \frac{d}{2m}$$

$$C_1 = x_0$$

$$C_2 = v_0 + \beta x_0$$

Spring Animations в iOS SDK UIView

```
extension UIView {

    class func animate(withDuration duration: TimeInterval,
                       delay: TimeInterval,
                       usingSpringWithDamping dampingRatio: CGFloat,
                       initialSpringVelocity velocity: CGFloat,
                       options: UIView.AnimationOptions = [],
                       animations: @escaping () -> Void,
                       completion: ((Bool) -> Void)? = nil)

}
```

Spring Animations в iOS SDK CASpringAnimation

```
open class CASpringAnimation : CABasicAnimation {

    /* The mass of the object attached to the end of the spring. Must be greater
       than 0. Defaults to one. */
    open var mass: CGFloat

    /* The spring stiffness coefficient. Must be greater than 0.
     * Defaults to 100. */
    open var stiffness: CGFloat


    /* The damping coefficient. Must be greater than or equal to 0.
     * Defaults to 10. */
    open var damping: CGFloat

}
```

Spring Animations в iOS SDK
CASpringAnimation

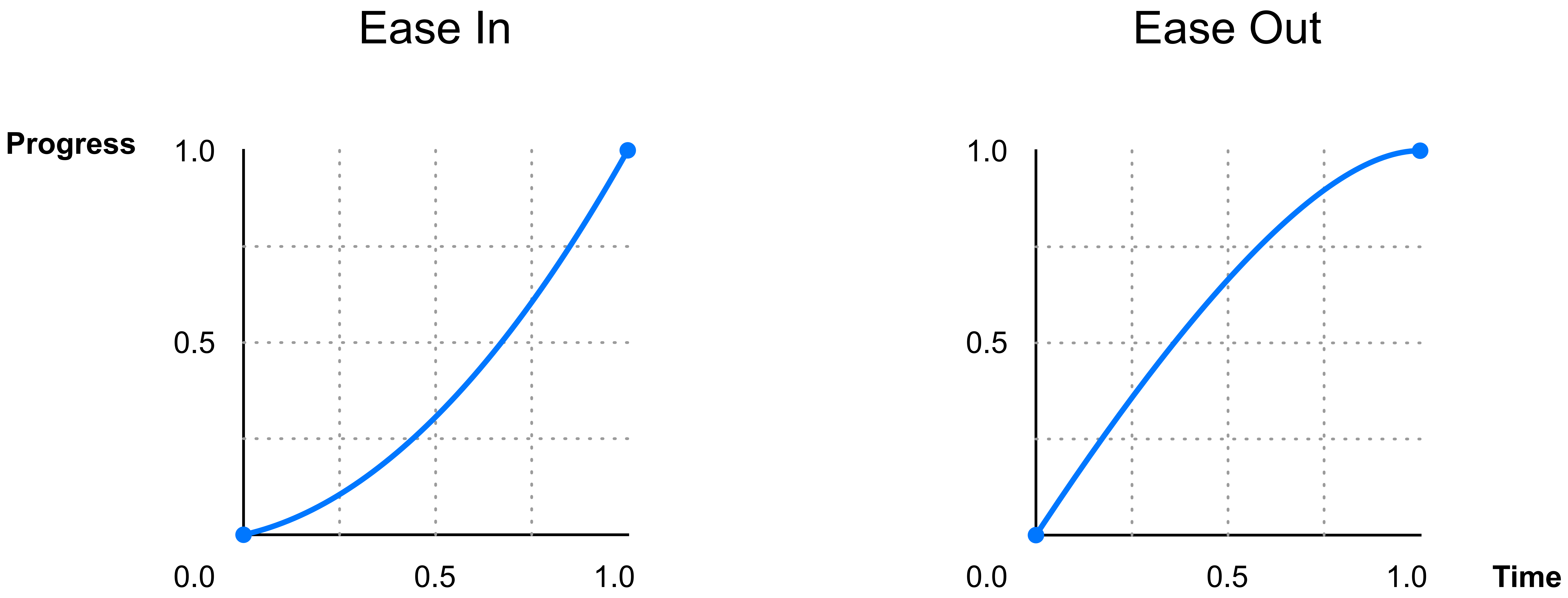
```
extension CASpringAnimation {

    convenience init(mass: CGFloat = 1, stiffness: CGFloat = 100,
                     dampingRatio: CGFloat)
    {
        self.init()

        self.mass = mass
        self.stiffness = stiffness
        self.damping = 2 * dampingRatio * sqrt(mass * stiffness)
    }

}
```

Spring Animations в iOS SDK CASpringAnimation

```
open class CASpringAnimation : CABasicAnimation {

    /* The basic duration of the object. Defaults to 0. */
    var duration: CFTimeInterval { get set }

    /* Returns the estimated duration required for the spring system to be
     * considered at rest. The duration is evaluated for the current animation
     * parameters. */
    open var settlingDuration: CFTimeInterval { get }

}
```

- **`settlingDuration`** не учитывает смещение пружины: **`fromValue`** и **`toValue`**

Spring Animations в iOS SDK

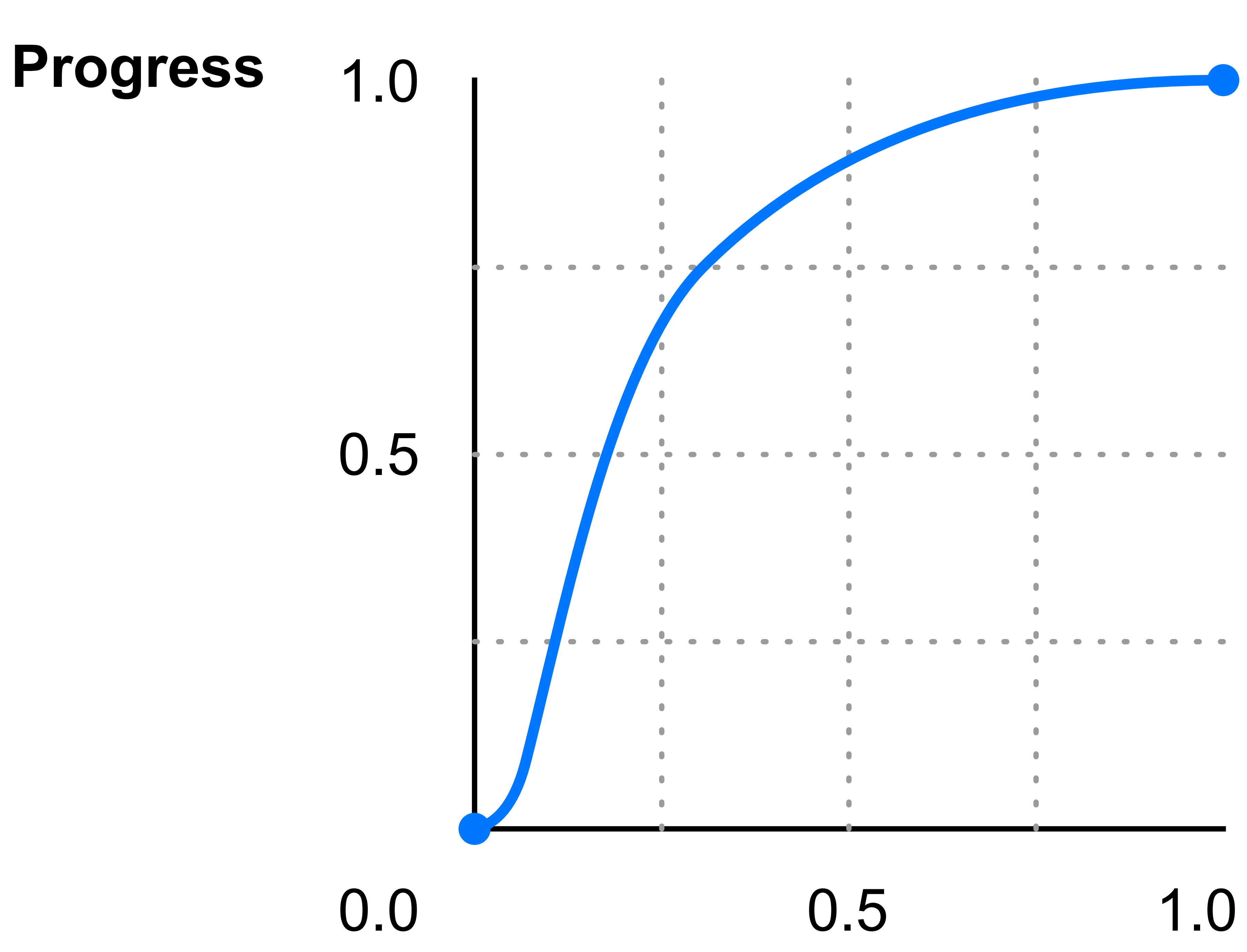


Advanced Animations with UIKit | developer.apple.com/videos/play/wwdc2017/230/

Spring Animations в iOS SDK

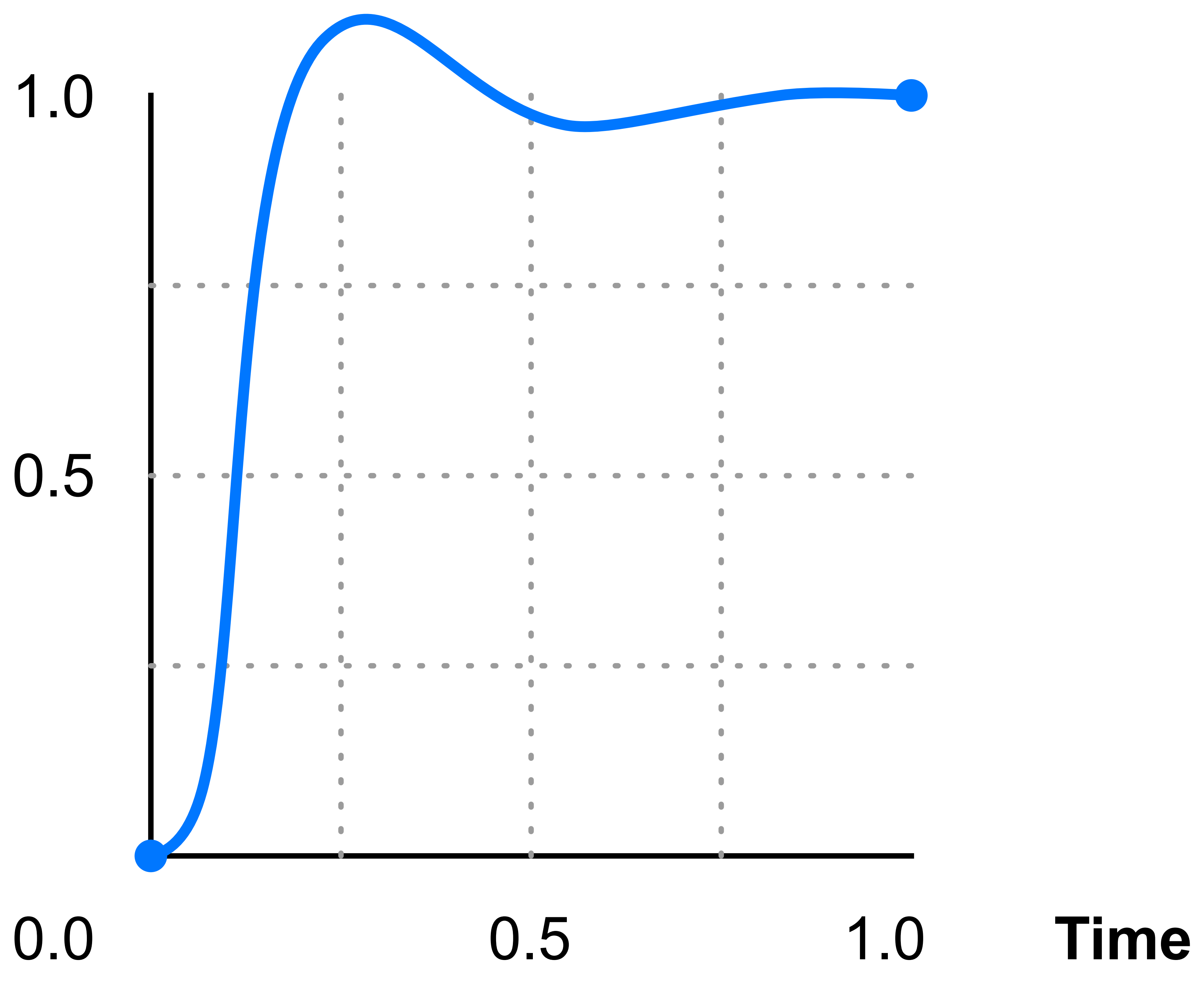
Critically Damped Spring

Damping ratio = 1.0



Underdamped Spring

Damping ratio < 1.0



Advanced Animations with UIKit | developer.apple.com/videos/play/wwdc2017/230/

Spring Animations в iOS SDK
UISpringTimingParameters

```
class UISpringTimingParameters : NSObject, UITimingCurveProvider {

    init(dampingRatio ratio: CGFloat, initialVelocity velocity: CGVector)

    init(mass: CGFloat, stiffness: CGFloat, damping: CGFloat,
        initialVelocity velocity: CGVector)

}
```

Spring Animations в iOS SDK
UISpringTimingParameters

```
let timingParameters = UISpringTimingParameters(mass: 1, stiffness: 100, damping: 10,
                                                initialVelocity: .zero)

let animator = UIViewPropertyAnimator(duration: 4, timingParameters: timingParameters)

animator.addAnimations { /* animations */ }
animator.startAnimation()

print(animator.duration) // 1.4727003346780927
```

Spring Animations в iOS SDK
UISpringTimingParameters

```
let timingParameters = UISpringTimingParameters(dampingRatio: 0.3,
                                                initialVelocity: .zero)

let animator = UIViewPropertyAnimator(duration: 4, timingParameters: timingParameters)

print(animator.duration) // 4.0
```

Spring Animations в iOS SDK
Нулевое смещение

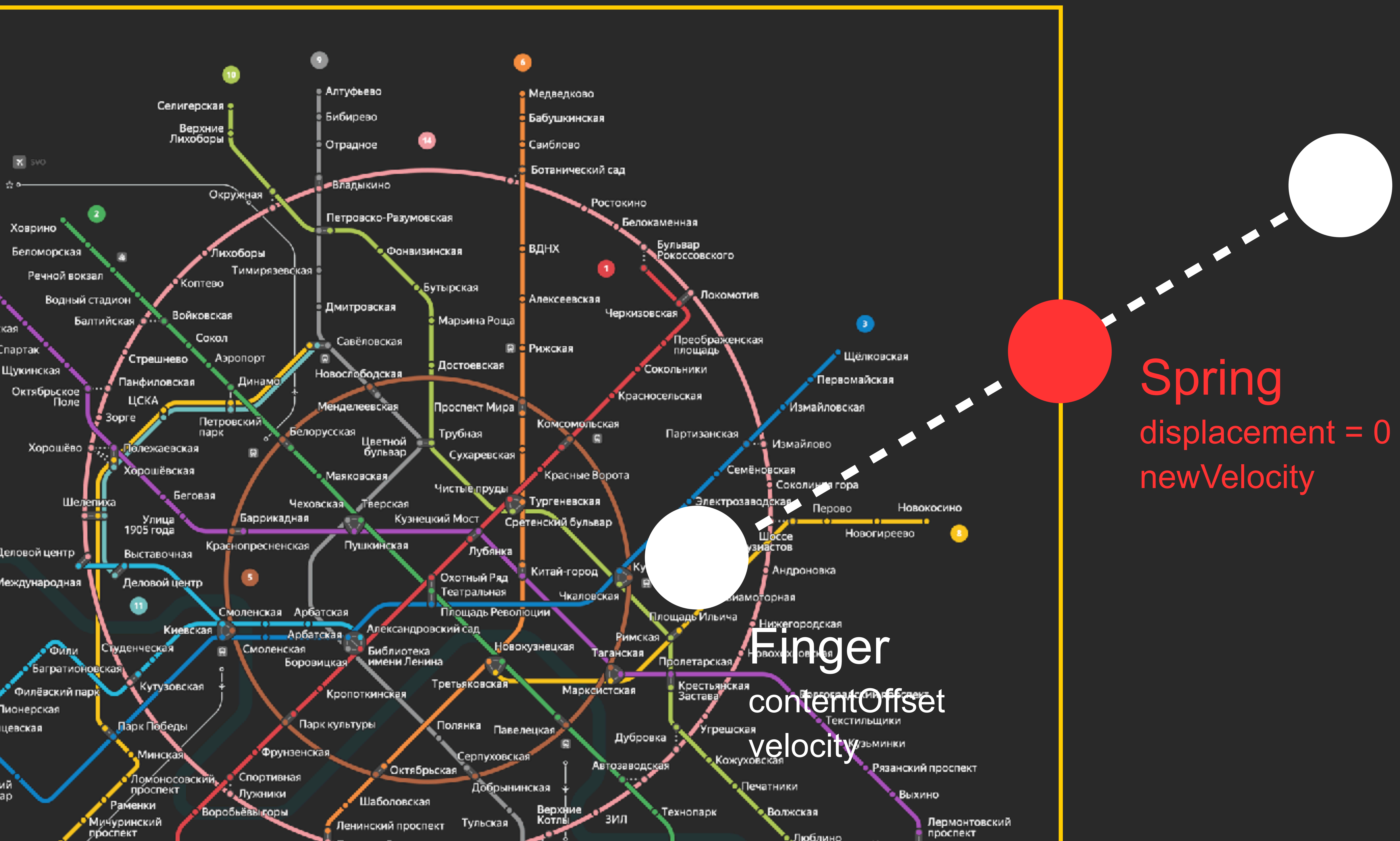
- Не работает с нулевым смещением: `fromValue == toValue`

```
UIView.animate(
    withDuration: 5,
    delay: 0,
    usingSpringWithDamping: 0.1,
    initialSpringVelocity: -1000,
    animations: {
        self.circle.frame = self.circle.frame
    })
```

SpringTimingParameters

```swift
struct Spring {
    var mass: CGFloat
    var stiffness: CGFloat
    var dampingRatio: CGFloat
}

struct SpringTimingParameters {
    var spring: Spring
    var displacement: CGPoint
    var initialVelocity: CGPoint
    var threshold: CGFloat
}

extension SpringTimingParameters {
    var duration: TimeInterval
    func value(at time: TimeInterval) -> CGPoint
}
```

Bounce



Scheme bounds

Destination

Spring

displacement = 0

newVelocity

Finger

contentOffset

velocity

DecelerationTimingParameters

```
extension DecelerationTimingParameters {

    func duration(to value: CGPoint) -> TimeInterval?

    func velocity(at time: TimeInterval) -> CGPoint

}
```

SimpleScrollView

```
@objc func handlePanRecognizer(_ sender: UIPanGestureRecognizer) {
    switch sender.state {
    case .began:
        state = .dragging(initialOffset: contentOffset)

    case .changed:
        let translation = sender.translation(in: self)
        if case .dragging(let initialOffset) = state {
            contentOffset = clampOffset(initialOffset - translation)
        }

    case .ended:
        state = .default
        let velocity = sender.velocity(in: self)
        startDeceleration(withVelocity: -velocity)

    // Other cases
    }
}
```

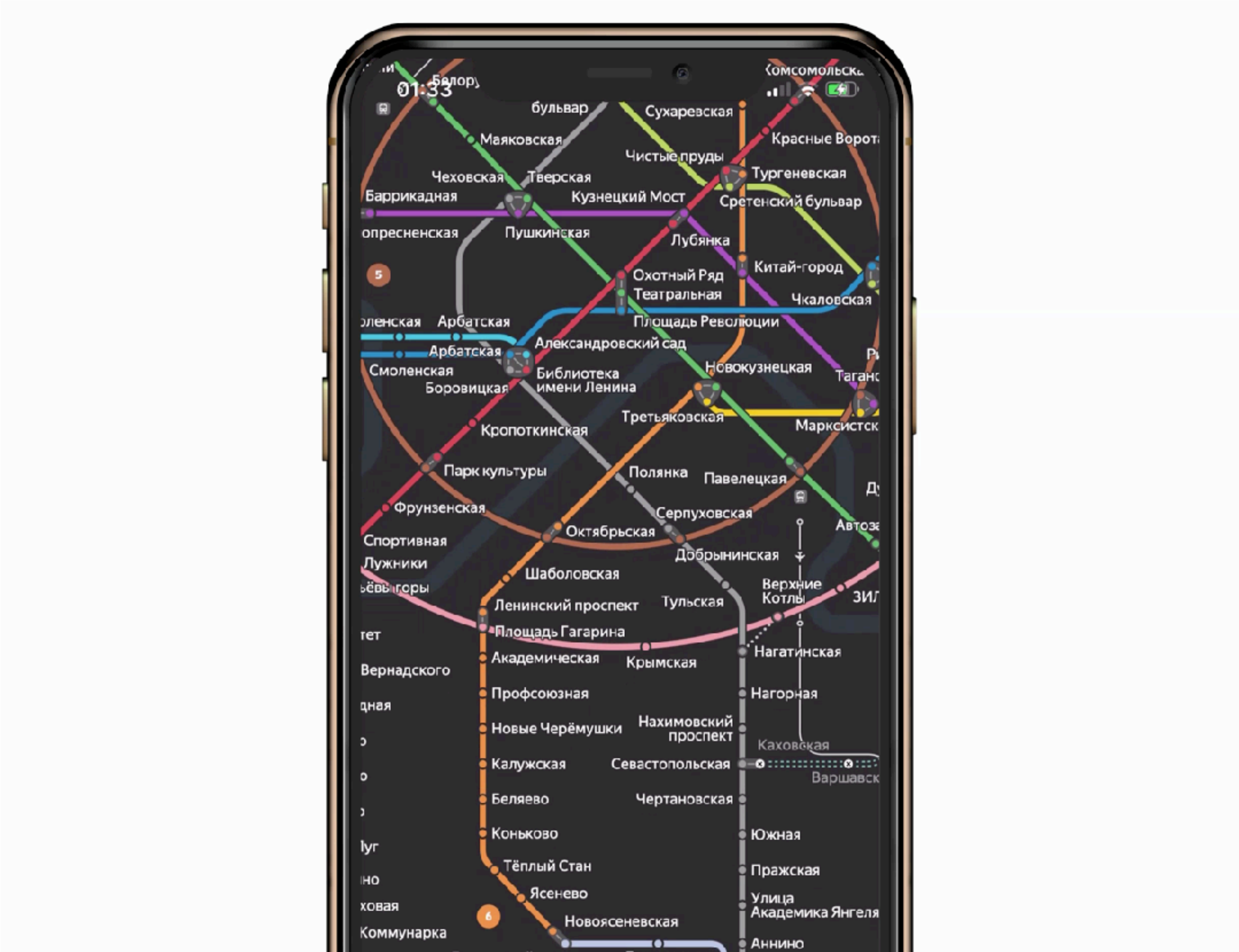
SimpleScrollView

```
func startDeceleration(withVelocity velocity: CGPoint) {
    let parameters = DecelerationTimingParameters(…)

    let destination = parameters.destination
    let intersection = getIntersection(rect: contentOffsetBounds, segment: (contentOffset, destination))

    let duration: TimeInterval
    if let intersection = intersection {
        duration = parameters.duration(to: intersection)
    } else {
        duration = parameters.duration
    }

    contentOffsetAnimation = TimerAnimation(
        duration: duration,
        animations: { [weak self] _, time in
            self?.contentOffset = parameters.value(at: time)
        },
        completion: { [weak self] finished in
            guard finished && intersection != nil else { return }
            let velocity = parameters.velocity(at: duration)
            self?.bounce(withVelocity: velocity)
        })
}
```

SimpleScrollView

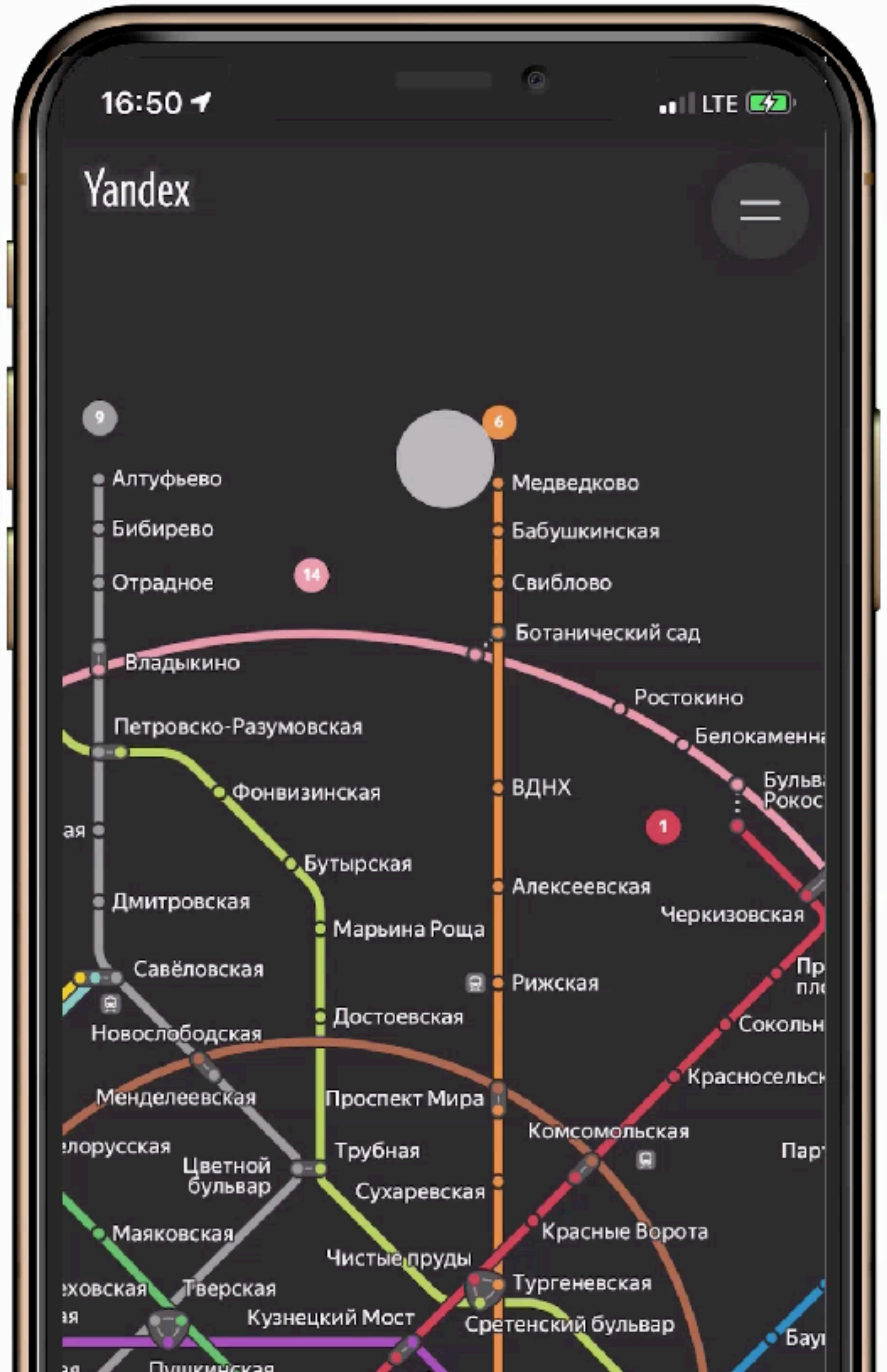
```
func bounce(withVelocity velocity: CGPoint) {
    let restOffset = contentOffset.clamped(to: contentOffsetBounds)
    let displacement = contentOffset - restOffset
    let threshold = 0.5 / UIScreen.main.scale
    let spring = Spring(mass: 1, stiffness: 100, dampingRatio: 1)

    let parameters = SpringTimingParameters(spring: spring,
                                            displacement: displacement,
                                            initialVelocity: velocity,
                                            threshold: threshold)


    contentOffsetAnimation = TimerAnimation(
        duration: parameters.duration,
        animations: { [weak self] _, time in
            self?.contentOffset = restOffset + parameters.value(at: time)
        })
}
```

Spring Animation

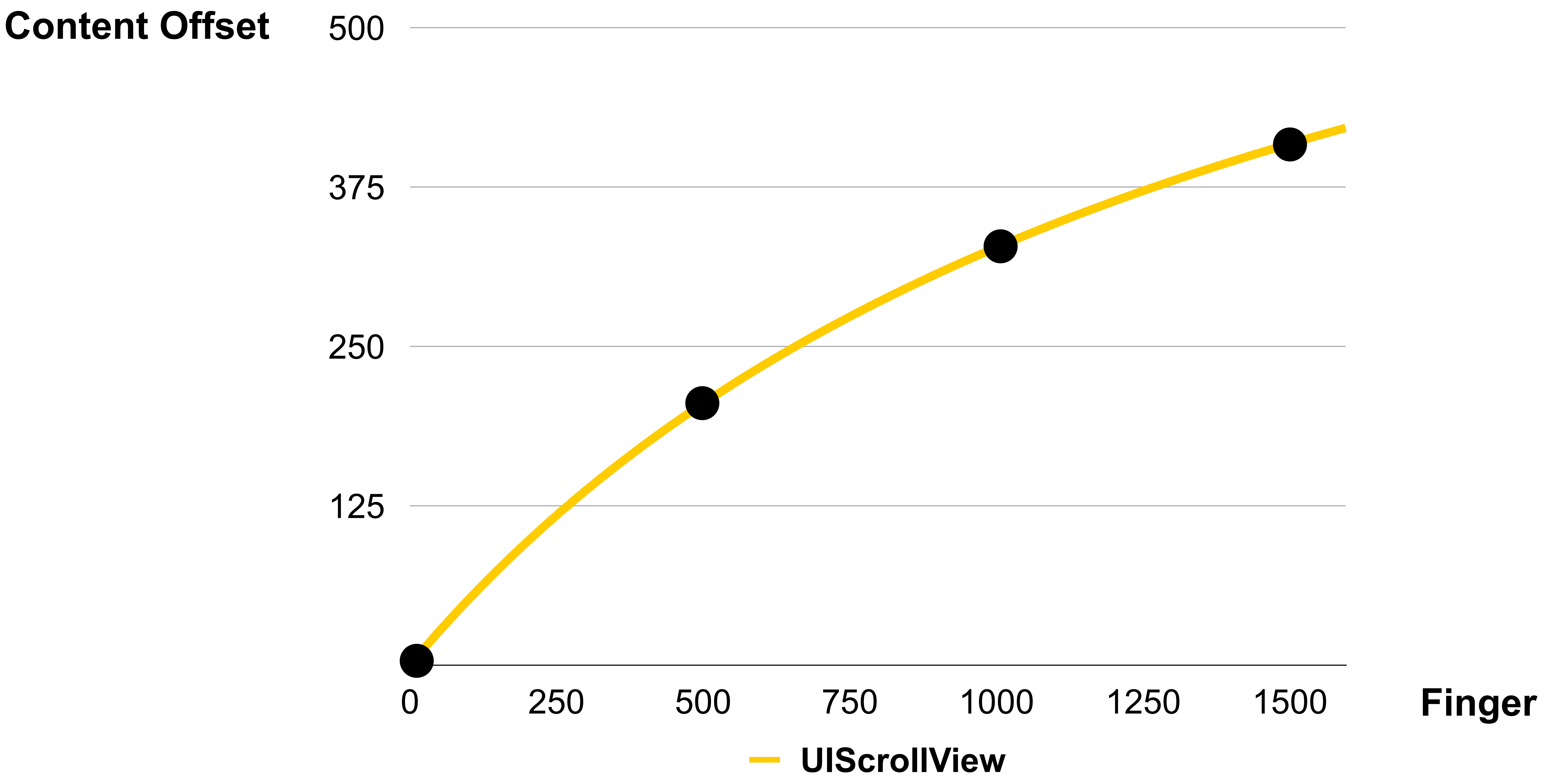


Rubber Band Effect

Rubber Band Effect



Rubber Band Effect



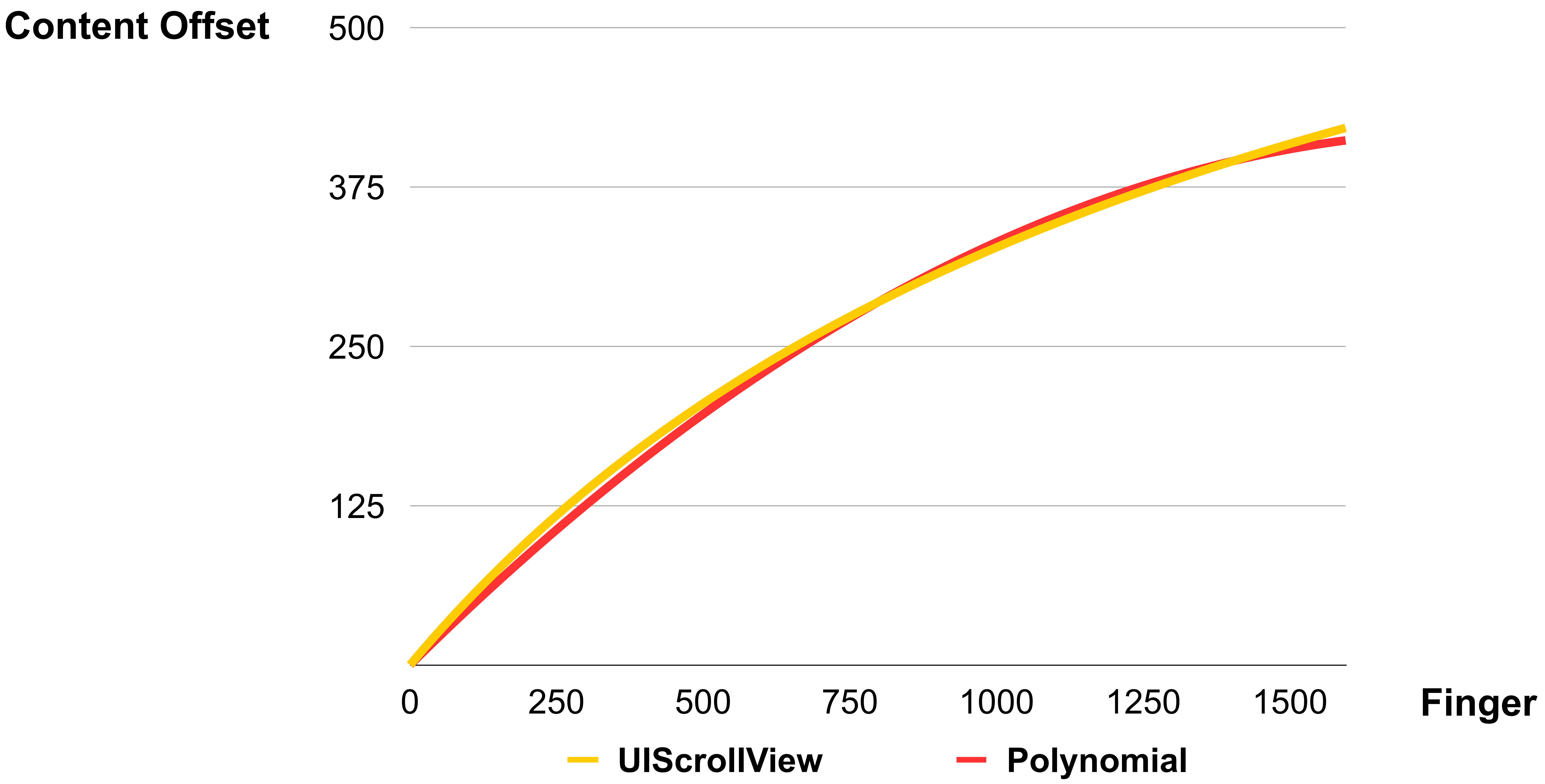
Rubber Band Effect
Полином

- wolframalpha.com

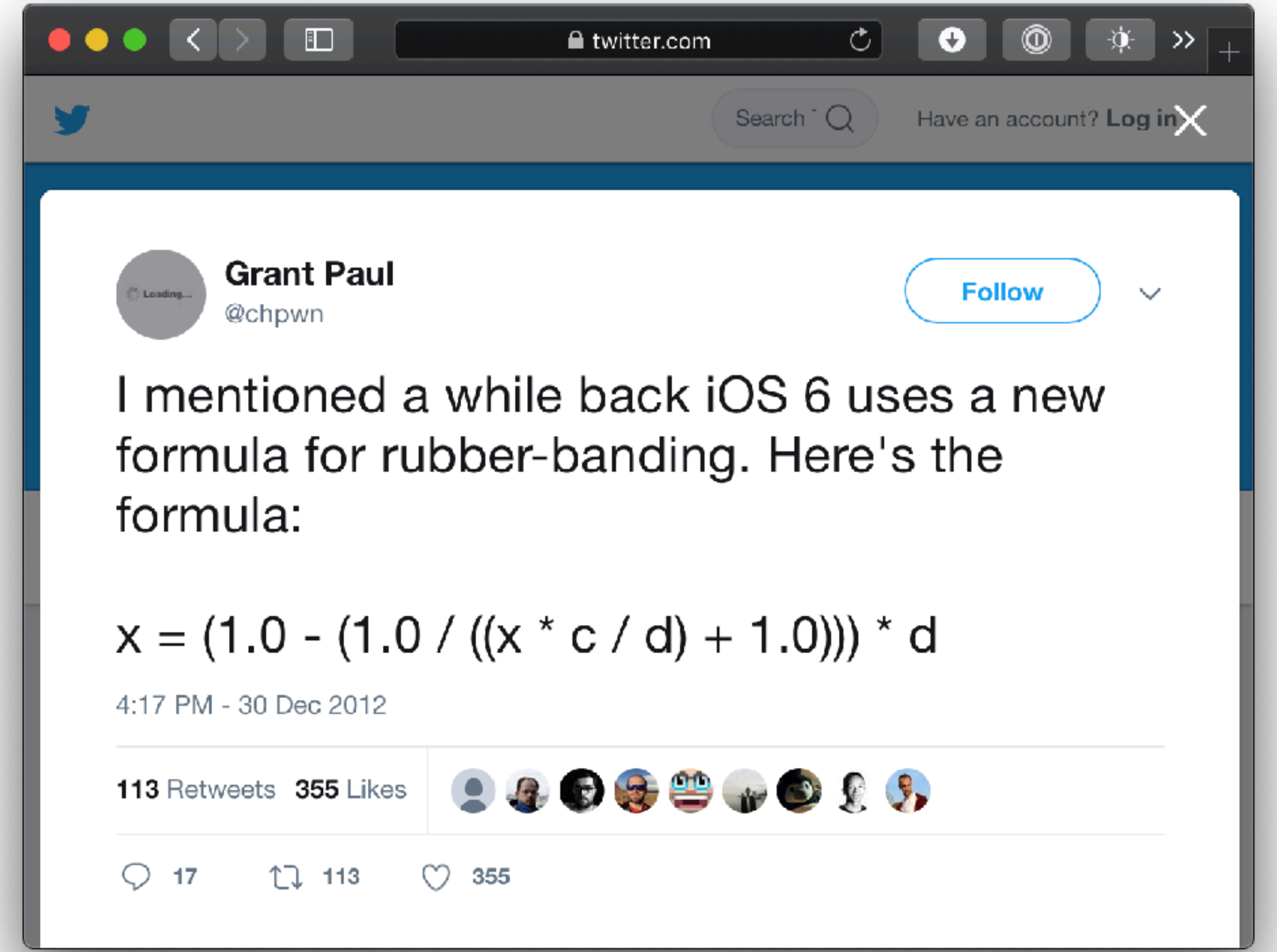
- quadratic fit {0, 0} {500, 205} {1000, 328} {1500, 409}

$$y = 2 + 0.456x - 0.000124x^2$$

Rubber Band Effect



Rubber Band Effect



Grant Paul
@chpwn

I mentioned a while back iOS 6 uses a new formula for rubber-banding. Here's the formula:

x = (1.0 - (1.0 / ((x * c / d) + 1.0))) * d

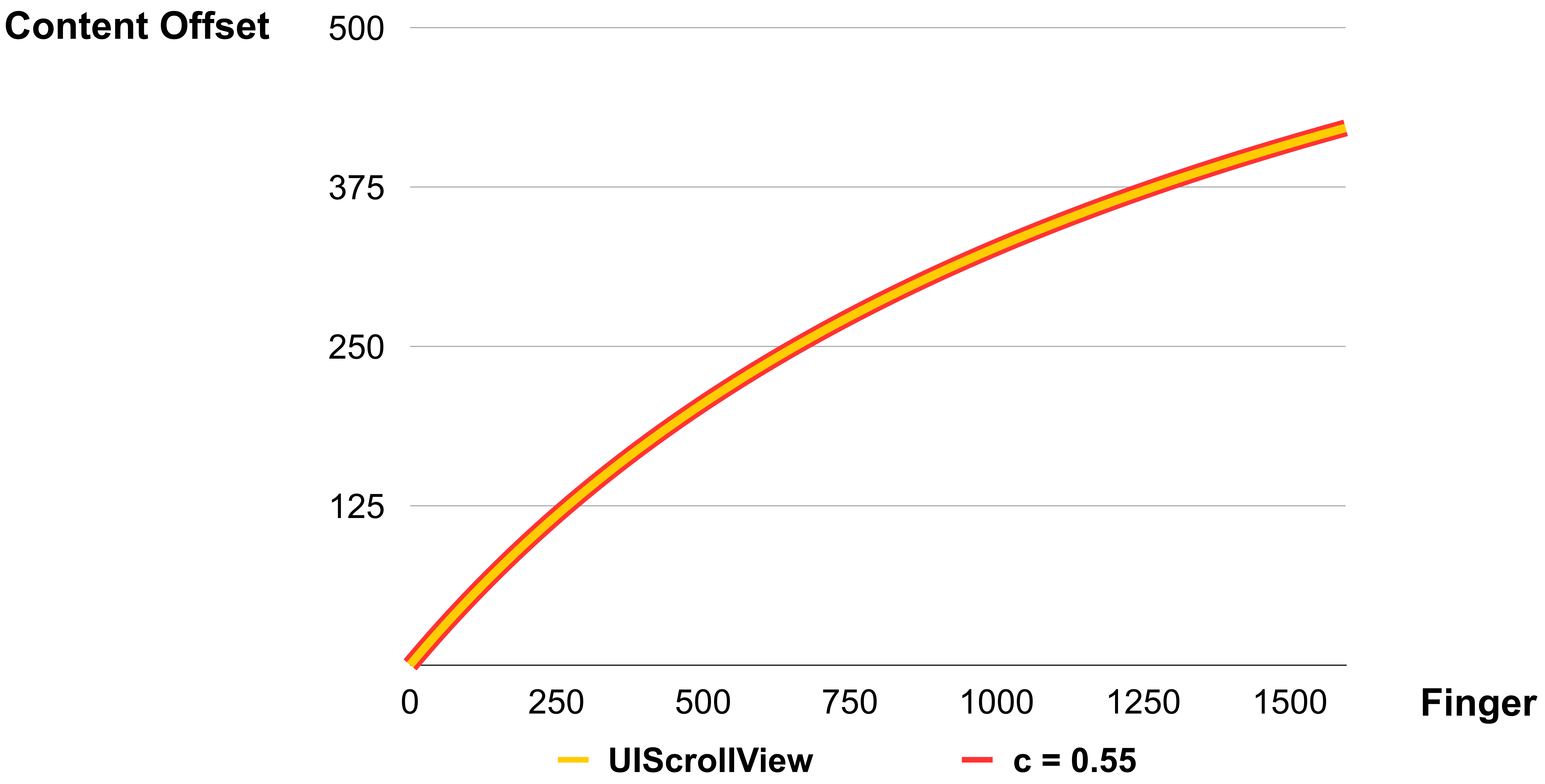
4:17 PM - 30 Dec 2012

113 Retweets 355 Likes

17 113 355

twitter.com/chpwn/status/285540192096497664

Rubber Band Effect

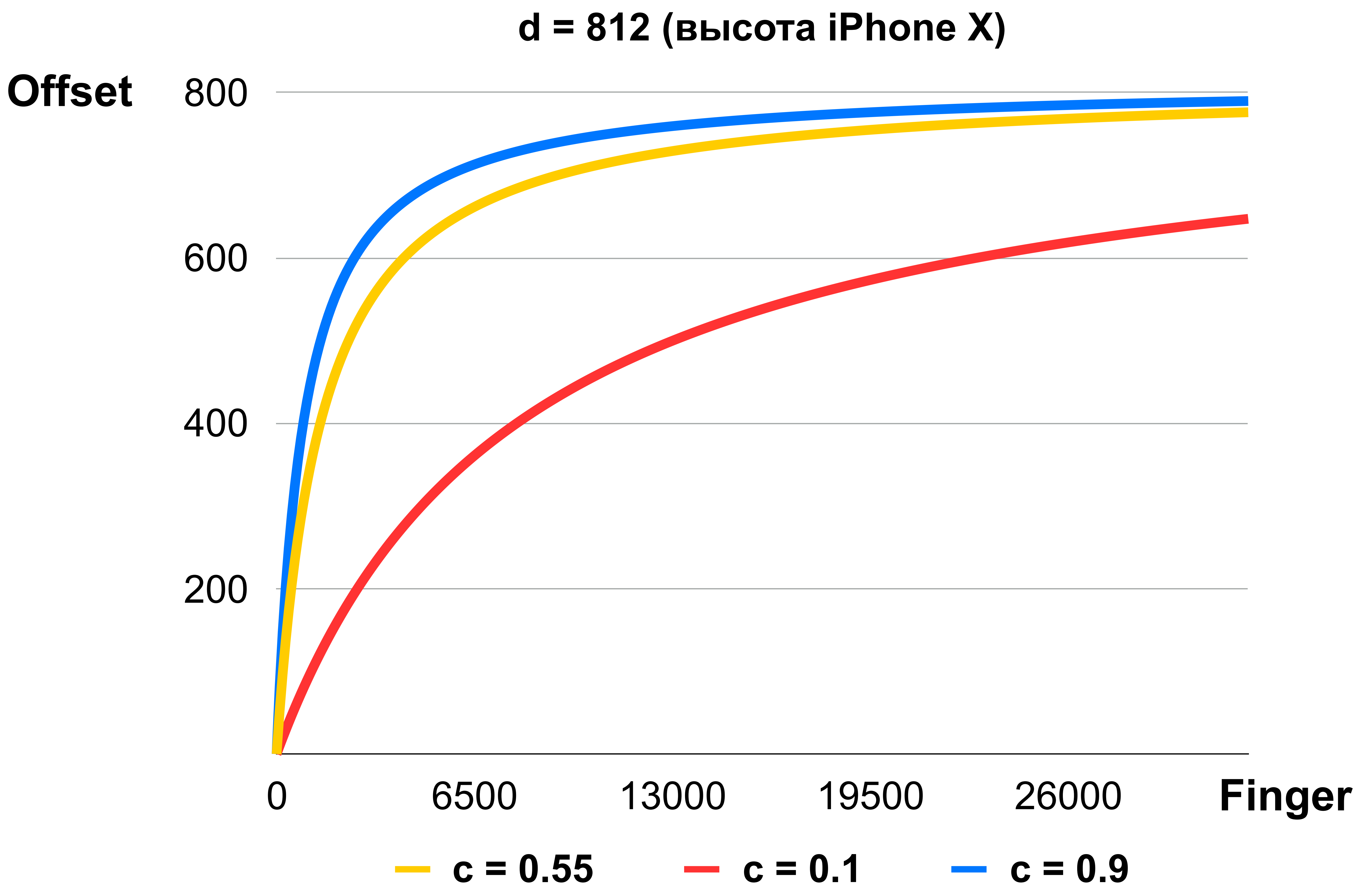


Rubber Band Effect

$$y = \left(1 - \frac{1}{\frac{cx}{d} + 1}\right)d$$

$$y \to d$$

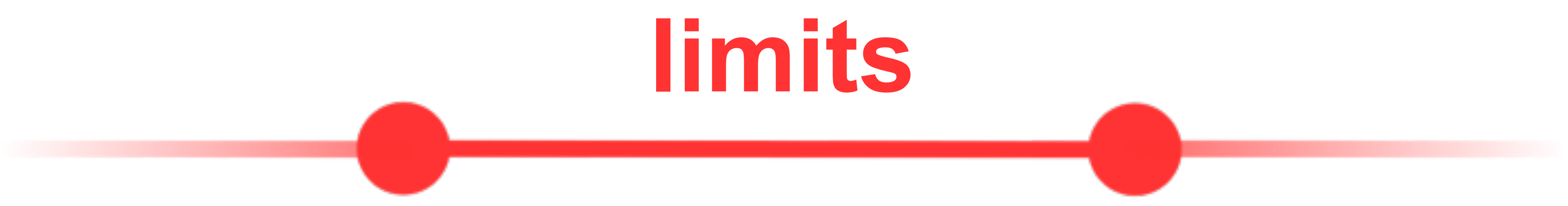
$$y < d$$



Rubber Band Effect

```
func rubberBandClamp(_ x: CGFloat, coeff: CGFloat, dim: CGFloat) -> CGFloat {
    return (1.0 - (1.0 / ((x * coeff / dim) + 1.0))) * dim
}
```

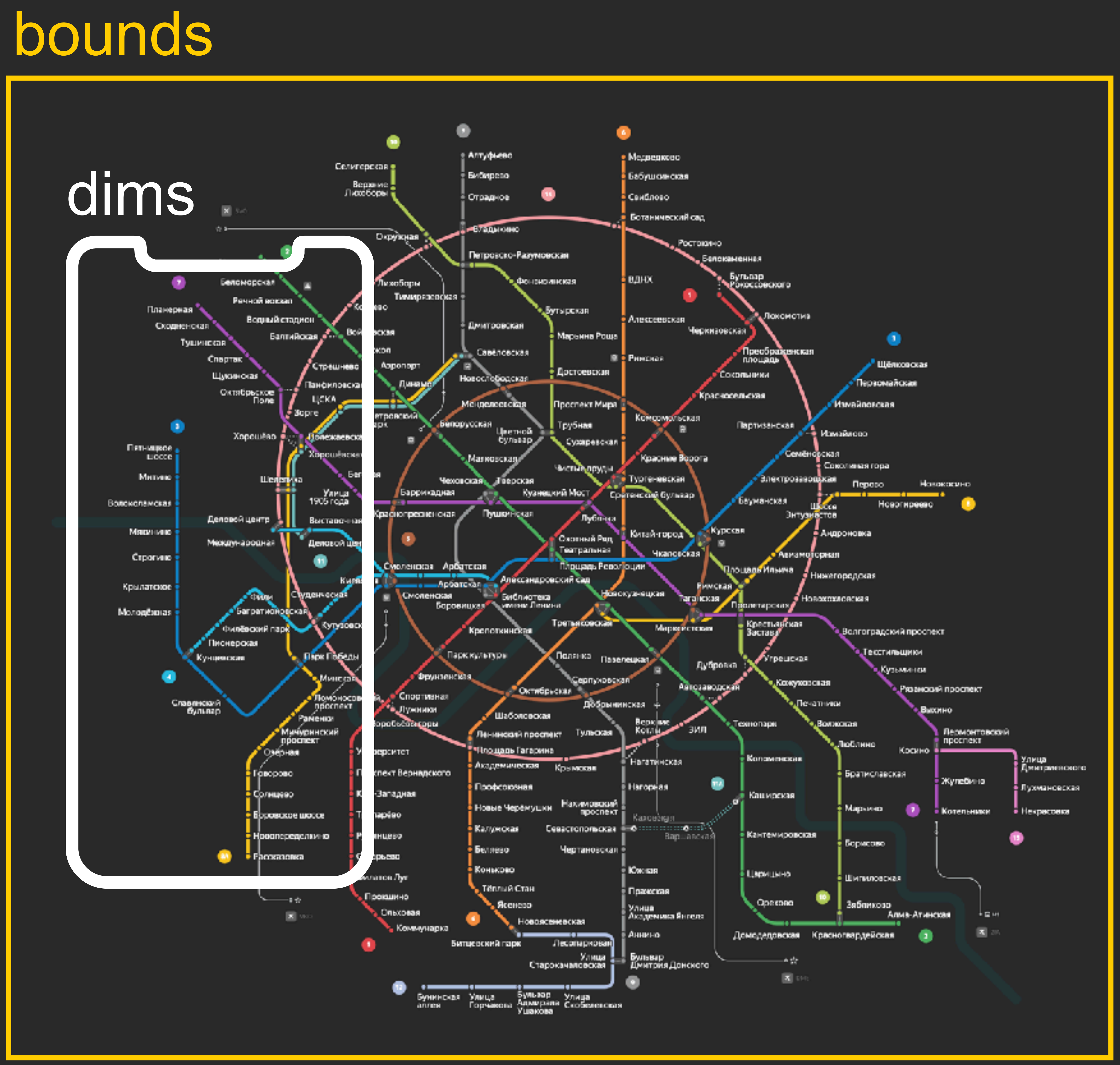
Rubber Band Effect



```
func rubberBandClamp(_ x: CGFloat, coeff: CGFloat, dim: CGFloat,
                     limits: ClosedRange<CGFloat>) -> CGFloat
{
    let clampedX = x.clamped(to: limits)
    let diff = abs(x - clampedX)
    let sign: CGFloat = clampedX > x ? -1 : 1

    return clampedX + sign * rubberBandClamp(diff, coeff: coeff, dim: dim)
}
```

Rubber Band Effect

```
struct RubberBand {
    var coeff: CGFloat = 0.55
    var dims: CGSize
    var bounds: CGRect

    func clamp(_ point: CGPoint) -> CGPoint
}
```



SimpleScrollView

```swift
@objc func handlePanRecognizer(_ sender: UIPanGestureRecognizer) {
    switch sender.state {
    case .began:
        state = .dragging(initialOffset: contentOffset)

    case .changed:
        let translation = sender.translation(in: self)
        if case .dragging(let initialOffset) = state {
            contentOffset = clampOffset(initialOffset - translation)
        }

    case .ended:
        state = .default
        let velocity = sender.velocity(in: self)
        startDeceleration(withVelocity: -velocity)

    // Other cases
    }
}
```

SimpleScrollView

```
func clampOffset(_ offset: CGPoint) -> CGPoint {
    return offset.clamped(to: contentOffsetBounds)
}
```

SimpleScrollView

```
func clampOffset(_ offset: CGPoint) -> CGPoint {
    let rubberBand = RubberBand(dims: frame.size, bounds: contentOffsetBounds)
    return rubberBand.clamp(offset)
}
```

SimpleScrollView

```
@objc func handlePanRecognizer(_ sender: UIPanGestureRecognizer) {
    switch sender.state {
    case .began:
        state = .dragging(initialOffset: contentOffset)

    case .changed:
        let translation = sender.translation(in: self)
        if case .dragging(let initialOffset) = state {
            contentOffset = clampOffset(initialOffset - translation)
        }

    case .ended:
        state = .default
        let velocity = sender.velocity(in: self)
        startDeceleration(withVelocity: -velocity)

    // Other cases
    }
}
```

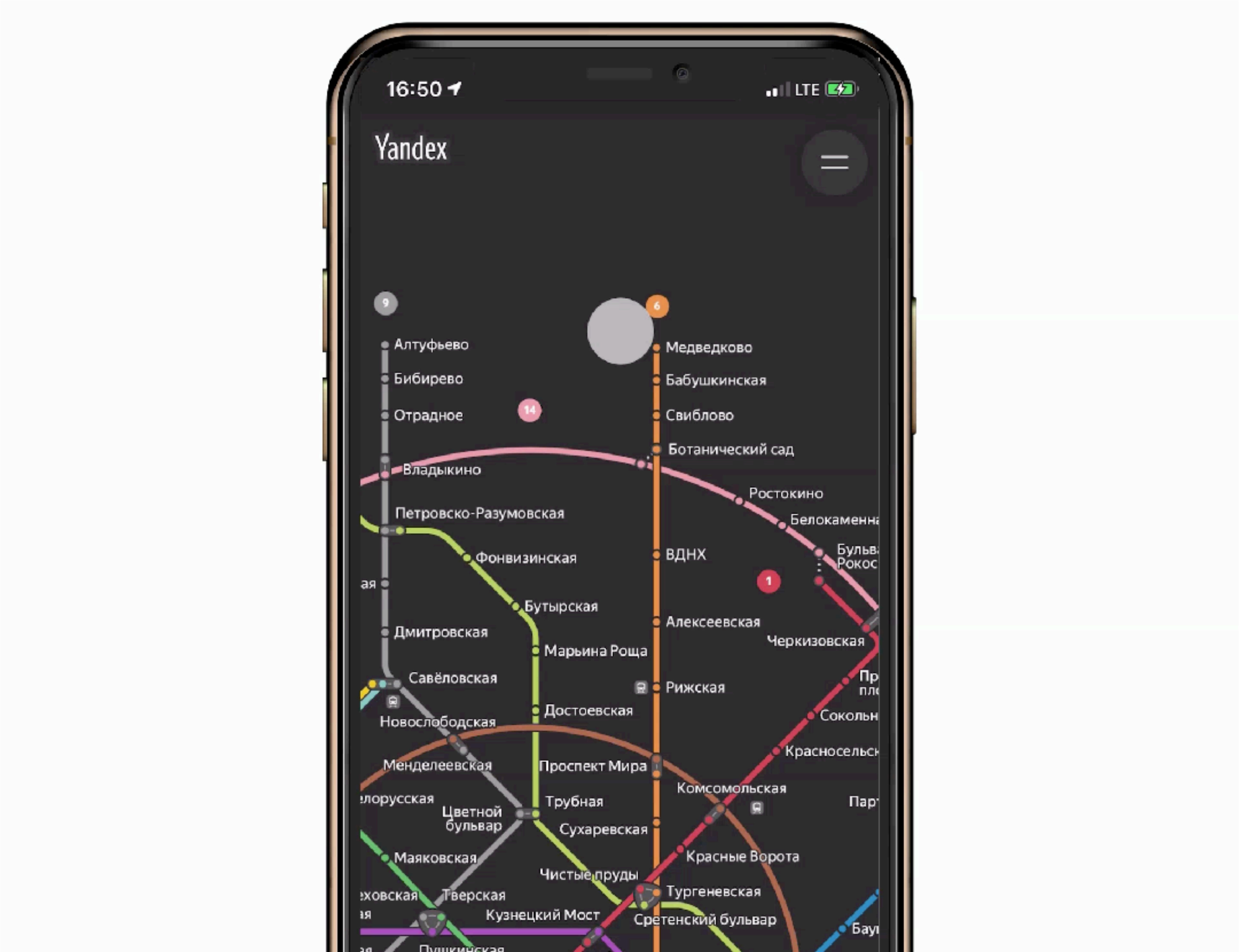
SimpleScrollView

```
@objc func handlePanRecognizer(_ sender: UIPanGestureRecognizer) {
    switch sender.state {
    case .began:
        state = .dragging(initialOffset: contentOffset)

    case .changed:
        let translation = sender.translation(in: self)
        if case .dragging(let initialOffset) = state {
            contentOffset = clampOffset(initialOffset - translation)
        }

    case .ended:
        state = .default
        let velocity = sender.velocity(in: self)
        completeGesture(withVelocity: -velocity)

    // Other cases
    }
}
```

SimpleScrollView

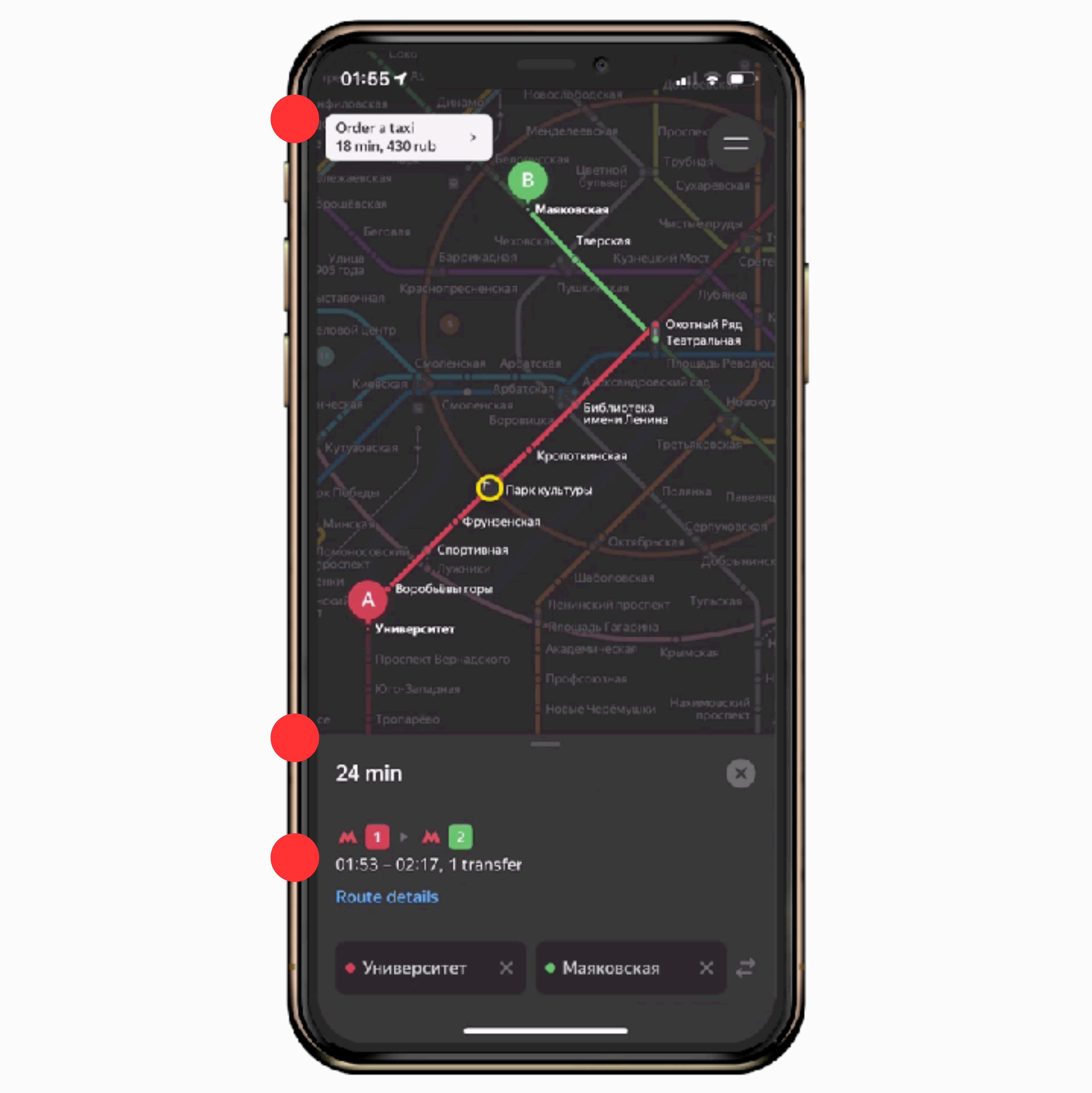
```
func completeGesture(withVelocity velocity: CGPoint) {
    if contentOffsetBounds.contains(contentOffset) {
        startDeceleration(withVelocity: velocity)
    } else {
        bounce(withVelocity: velocity)
    }
}
```

Rubber Band Effect



Примеры

Карточка
Переход между состояниями



01:55

Order a taxi
18 min, 430 rub

24 min

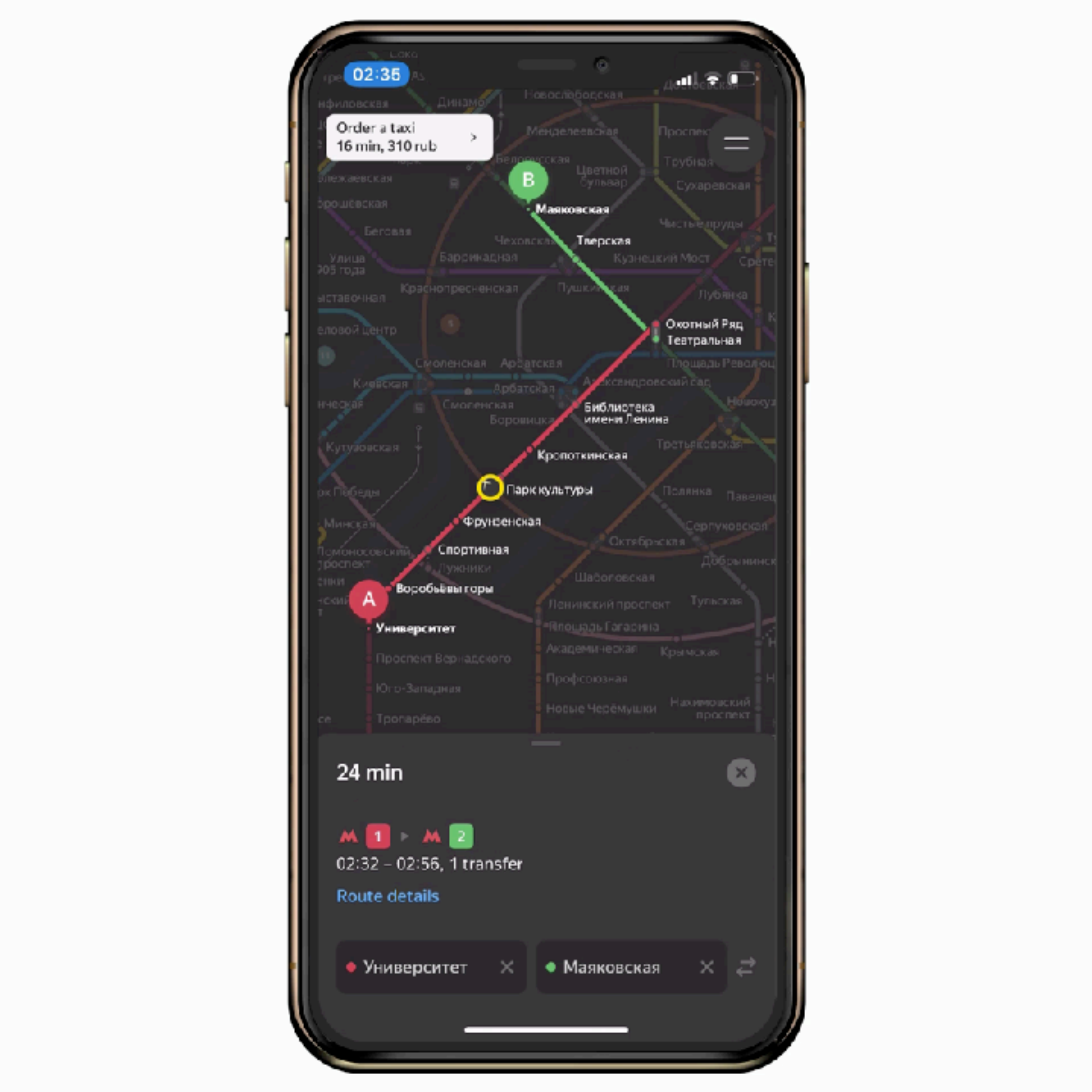
01:53 – 02:17, 1 transfer

Route details

Университет

Маяковская

Карточка
Переход между состояниями



Order a taxi
16 min, 310 rub

24 min

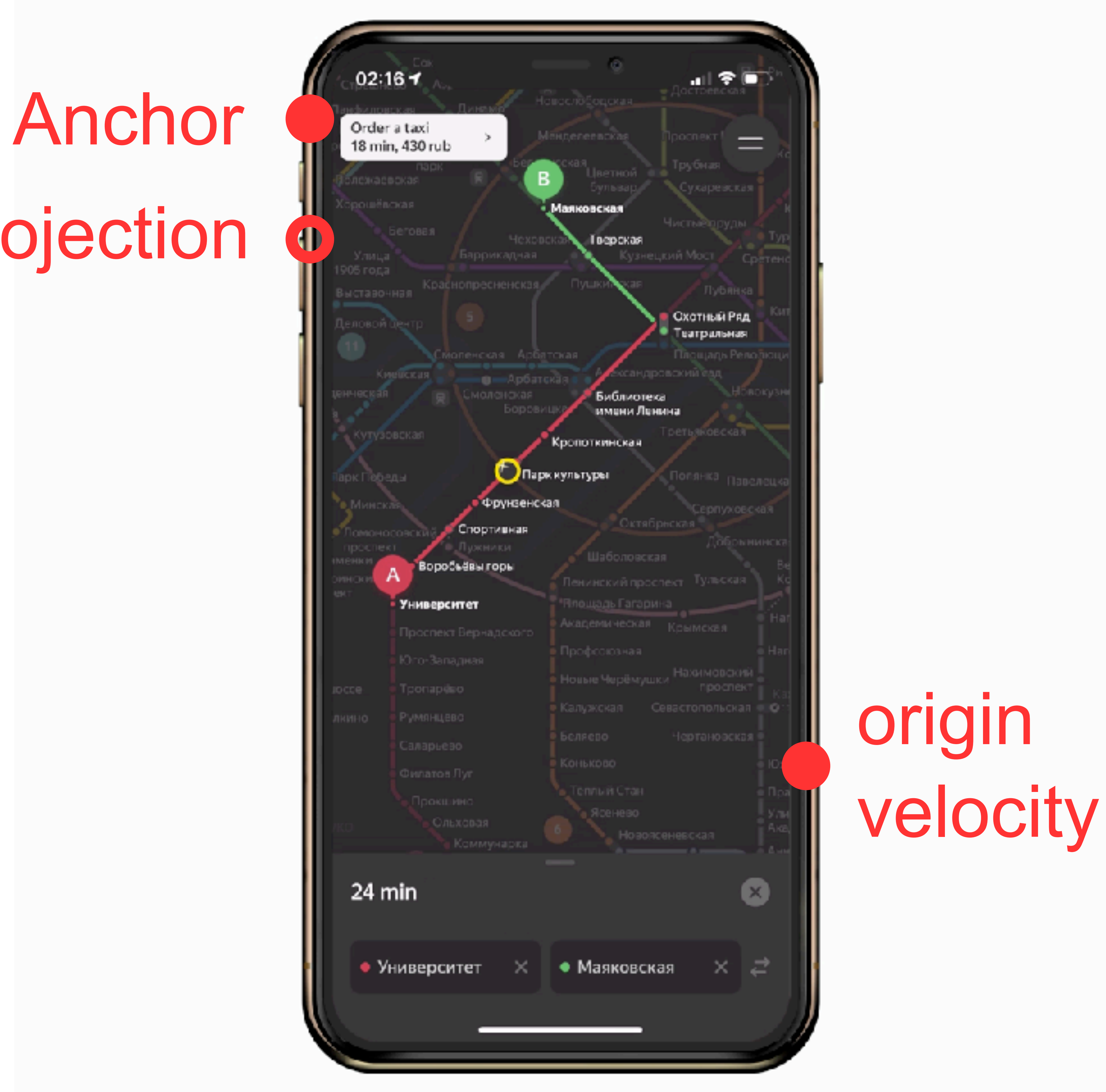
02:32 – 02:56, 1 transfer

Route details

● Университет ✕

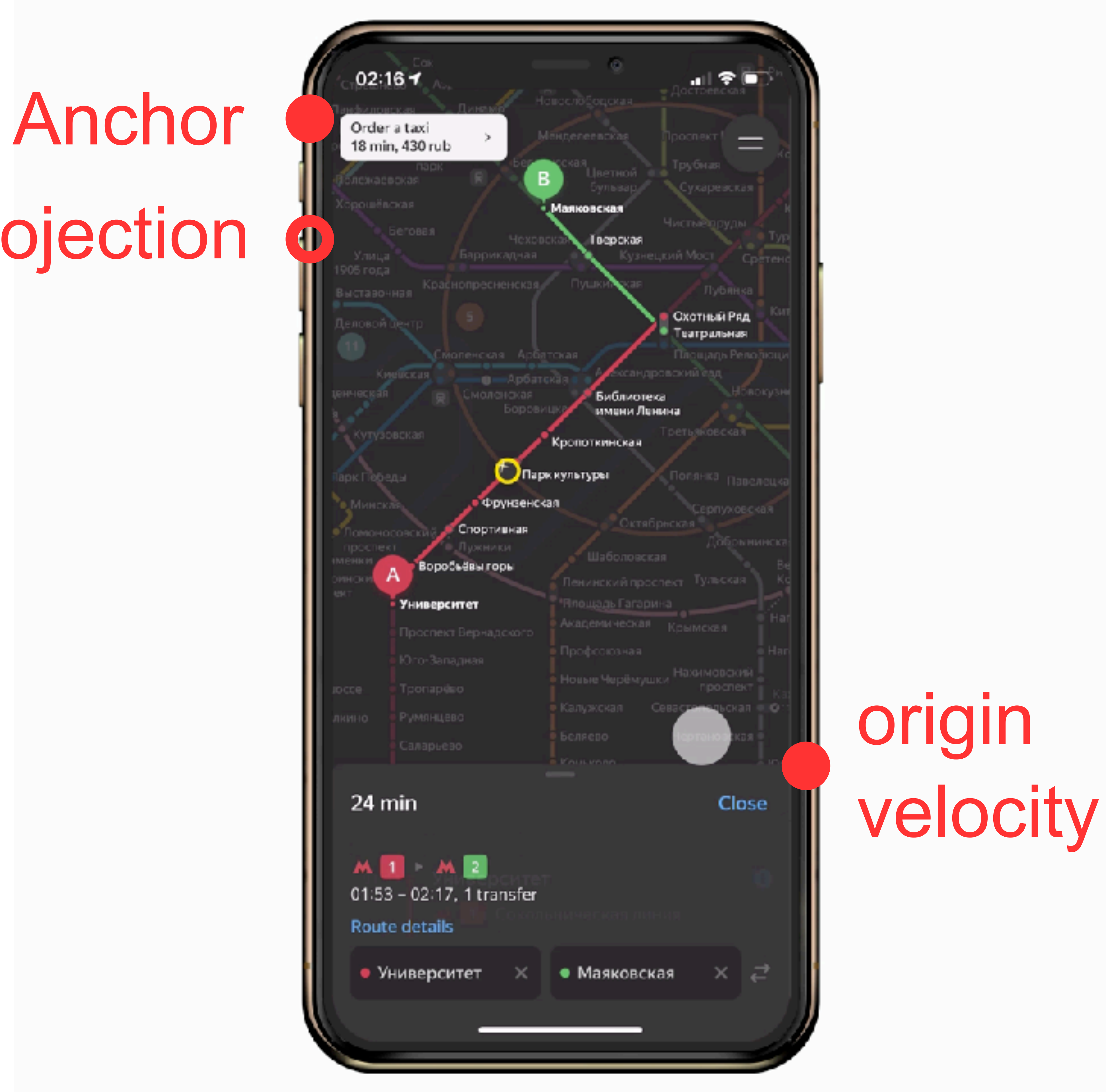
● Маяковская ✕

Карточка
Переход между состояниями



Designing Fluid Interfaces | developer.apple.com/videos/play/wwdc2018/803/

Карточка
Переход между состояниями



Anchor

Projection

origin
velocity

Order a taxi
18 min, 430 rub

24 min

Close

1 ▸ 2

01:53 – 02:17, 1 transfer

Route details

● Университет ✕

● Маяковская ✕

Designing Fluid Interfaces | developer.apple.com/videos/play/wwdc2018/803/

Карточка
Поиск проекции

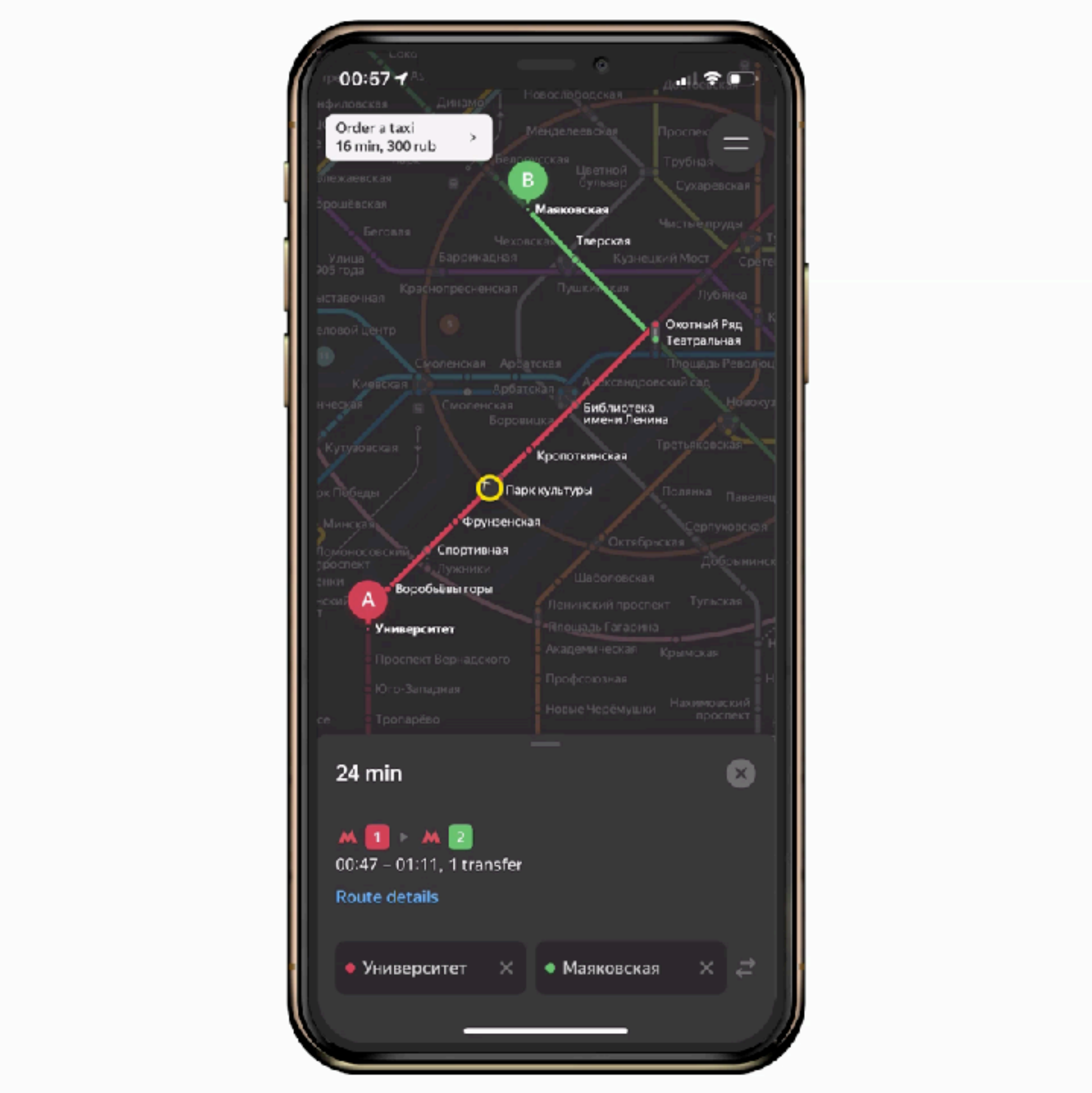
$$X = x_0 - \frac{v_0}{1000 ln(d)}$$

```
func project(value: CGPoint, velocity: CGPoint, decelerationRate: CGFloat) -> CGPoint {
    return value - velocity / (1000.0 * log(decelerationRate))
}
```

Карточка

```swift
func completeGesture(velocity: CGPoint) {
    let decelerationRate = UIScrollView.DecelerationRate.normal.rawValue

    let projection = project(value: self.origin, velocity: velocity,
                             decelerationRate: decelerationRate)

    let anchor = nearestAnchor(to: projection)

    UIView.animate(withDuration: 0.25) { [weak self] in
        self?.origin = anchor
    }
}
```

Карточка
Переход между состояниями



Order a taxi
16 min, 300 rub

24 min

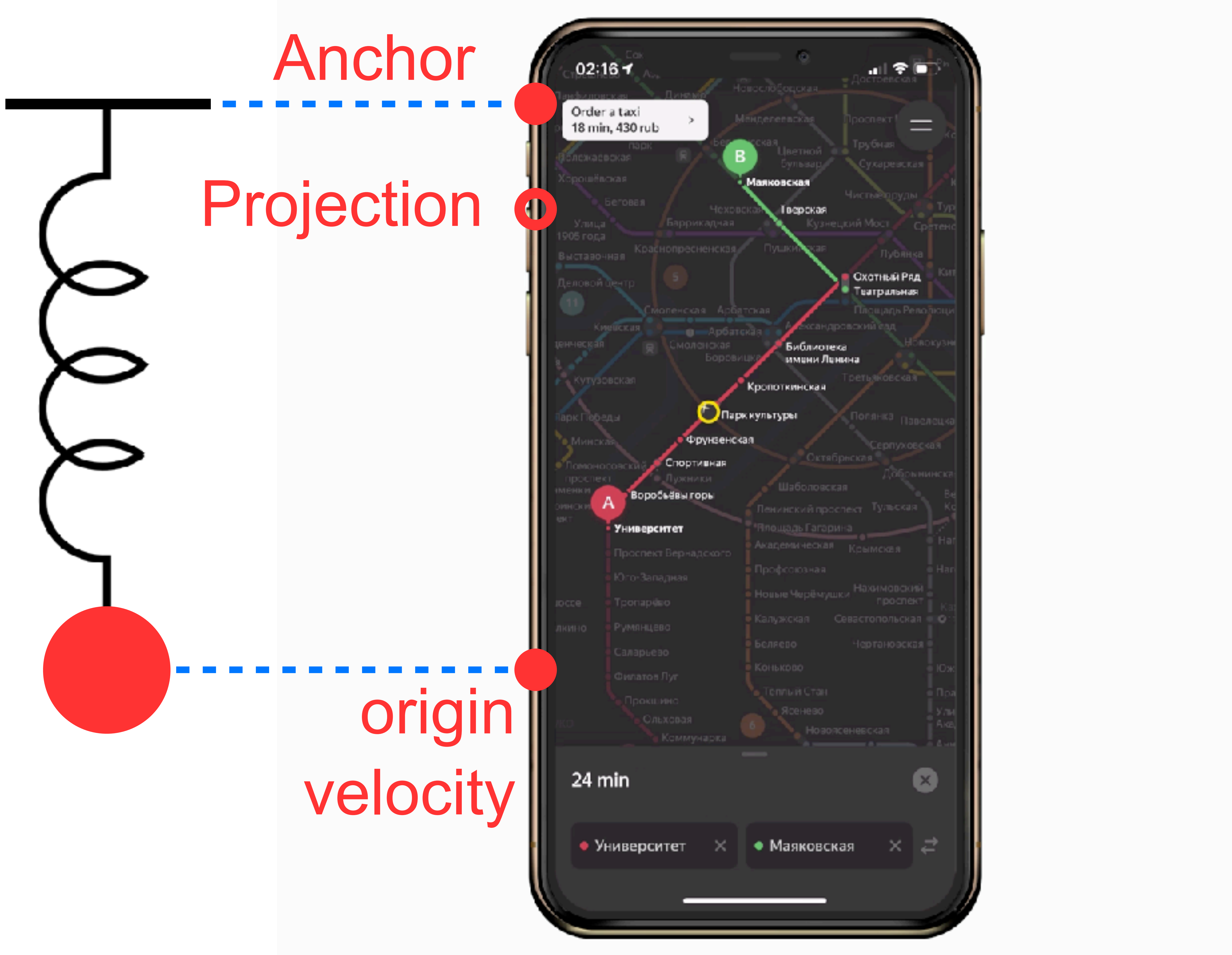
00:47 – 01:11, 1 transfer

Route details

Университет

Маяковская

Карточка
Переход между состояниями

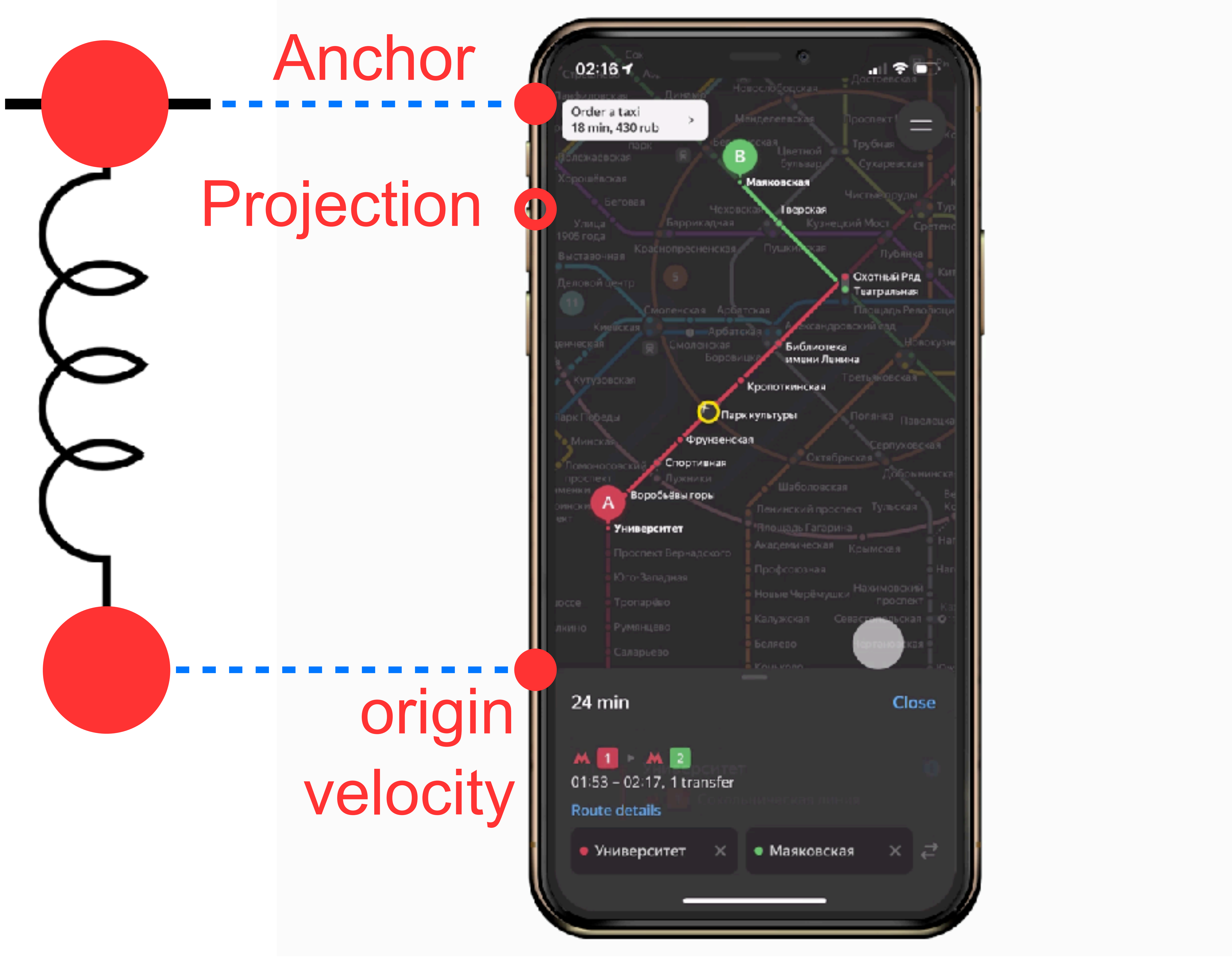


Order a taxi
18 min, 430 rub

24 min

Университет ✕ | Маяковская ✕

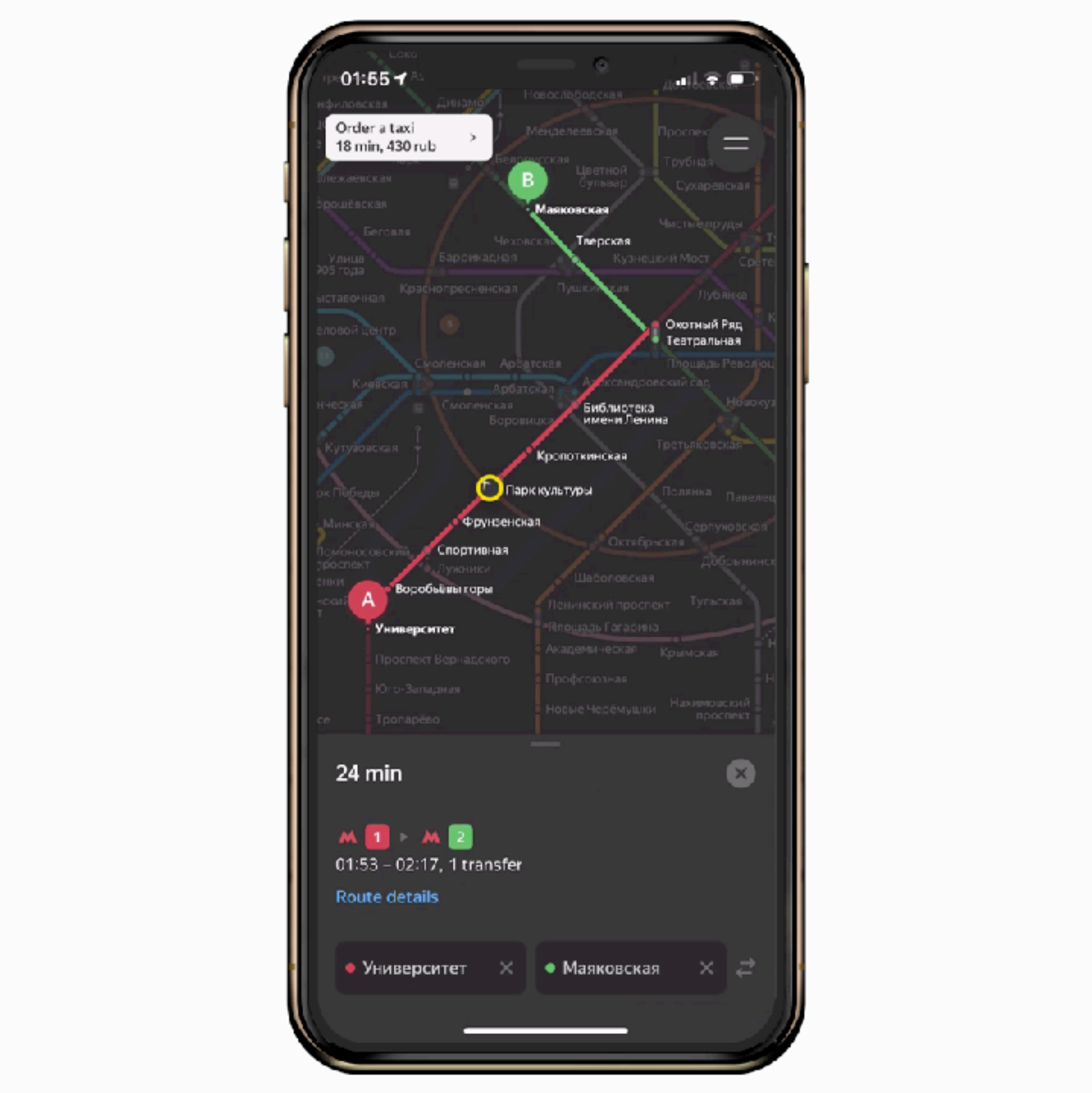
Карточка
Переход между состояниями



Карточка

```
func completeGesture(velocity: CGPoint) {
    let decelerationRate = UIScrollView.DecelerationRate.normal.rawValue
    let projection = project(value: self.origin, velocity: velocity,
                             decelerationRate: decelerationRate)

    let anchor = nearestAnchor(to: projection)

    let timingParameters = SpringTimingParameters(
        spring: Spring(mass: 1, stiffness: 200, dampingRatio: 1),
        displacement: self.origin - anchor,
        initialVelocity: velocity,
        threshold: 0.5 / UIScreen.main.scale)

    originAnimation = TimerAnimation(
        duration: timingParameters.duration,
        animations: { [weak self] _, time in
            self?.origin = anchor + timingParameters.value(at: time)
        })
}
```

Карточка
Damping Ratio = 1.0



Order a taxi
18 min, 430 rub

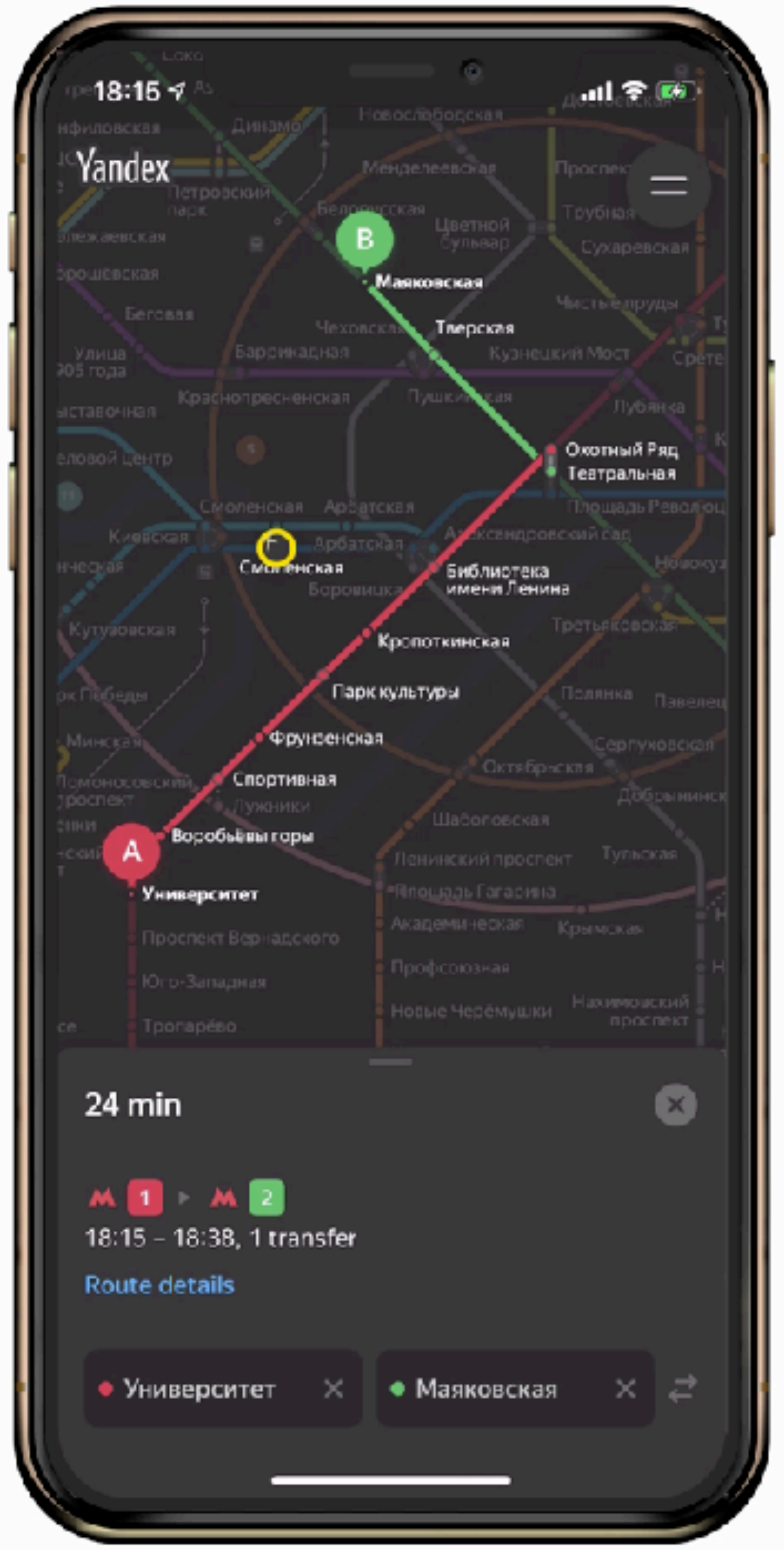
24 min

1 ▸ 2

01:53 – 02:17, 1 transfer

Route details

Карточка
Damping Ratio < 1.0



24 min

1 ▸ 2

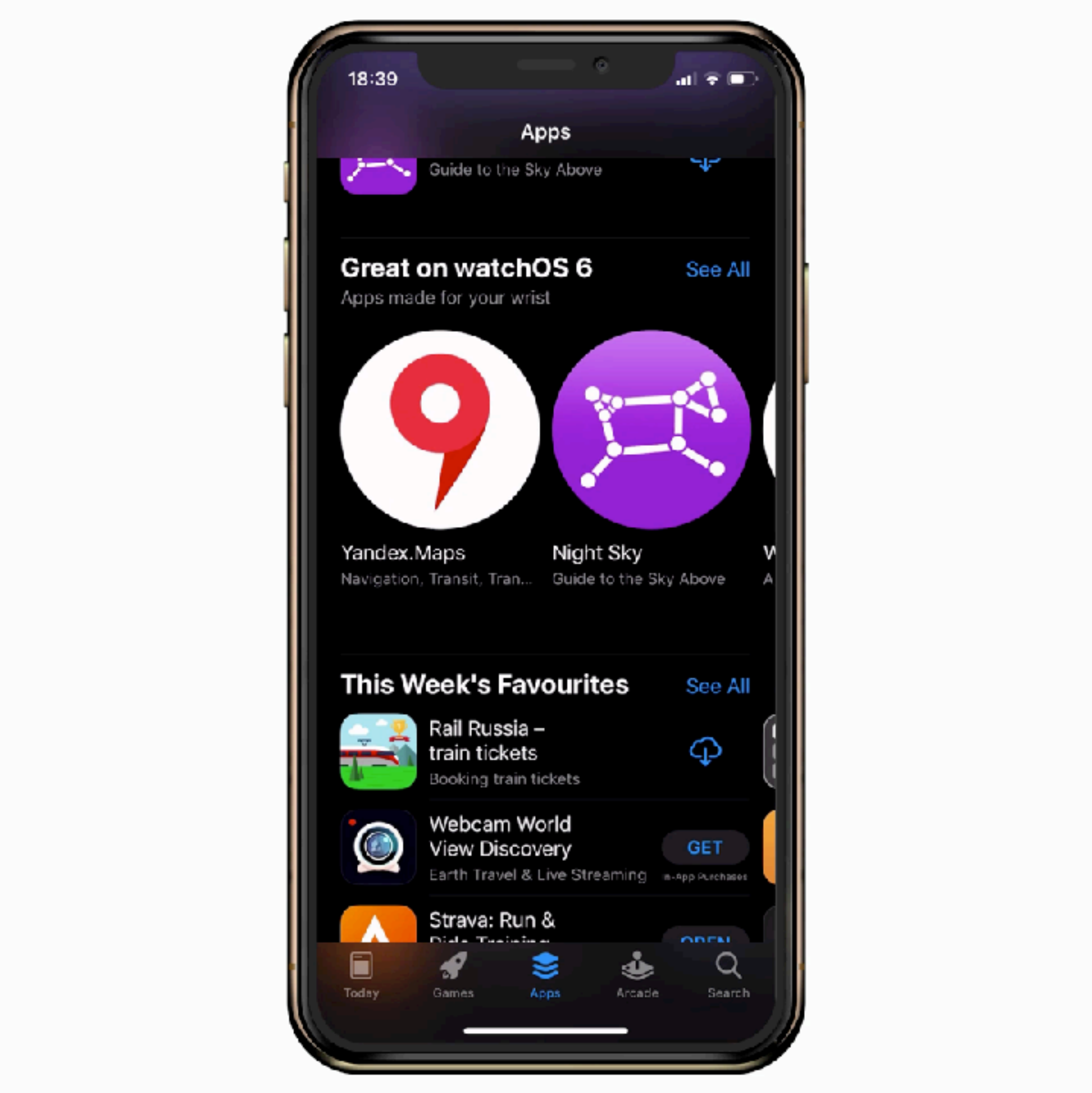
18:15 – 18:38, 1 transfer

Route details

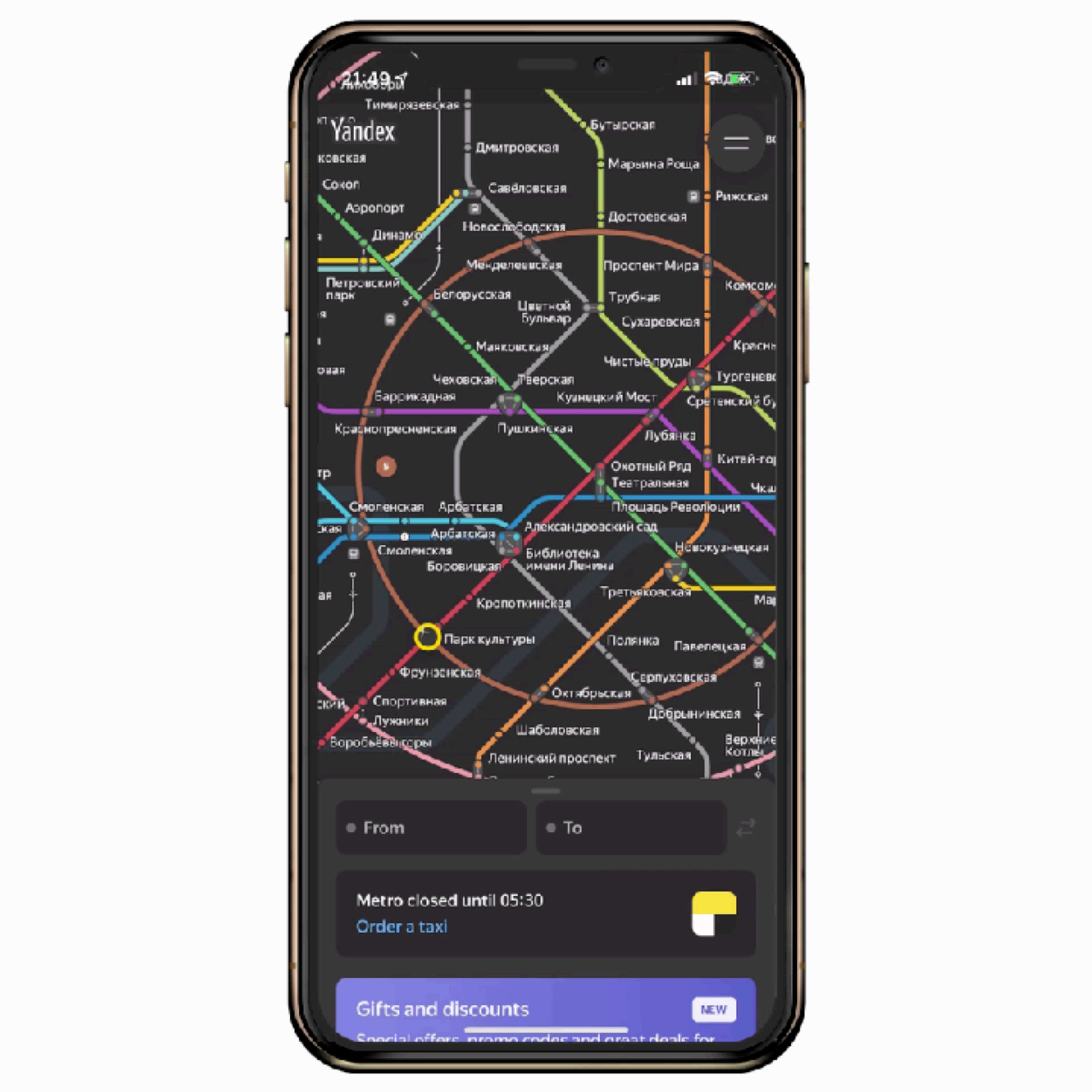
Университет ✕ | Маяковская ✕

Picture in Picture

Кастомная пейджинация
App Store



Карточка
Rubber Band Effect



From

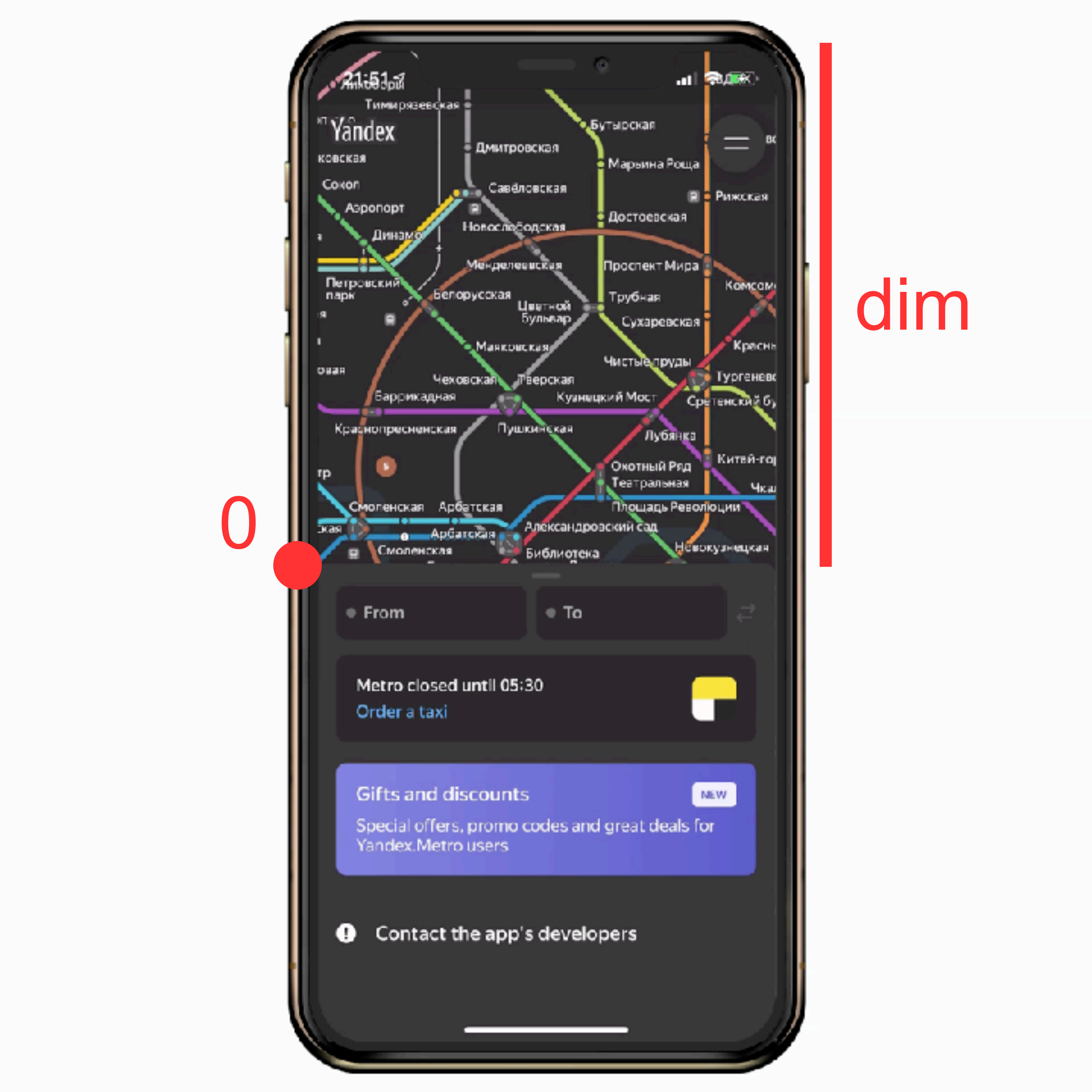
To

Metro closed until 05:30

Order a taxi

Gifts and discounts NEW

Карточка
Rubber Band Effect



Metro closed until 05:30

Order a taxi

Gifts and discounts

Special offers, promo codes and great deals for Yandex.Metro users

Contact the app's developers

Карточка
Rubber Band Effect

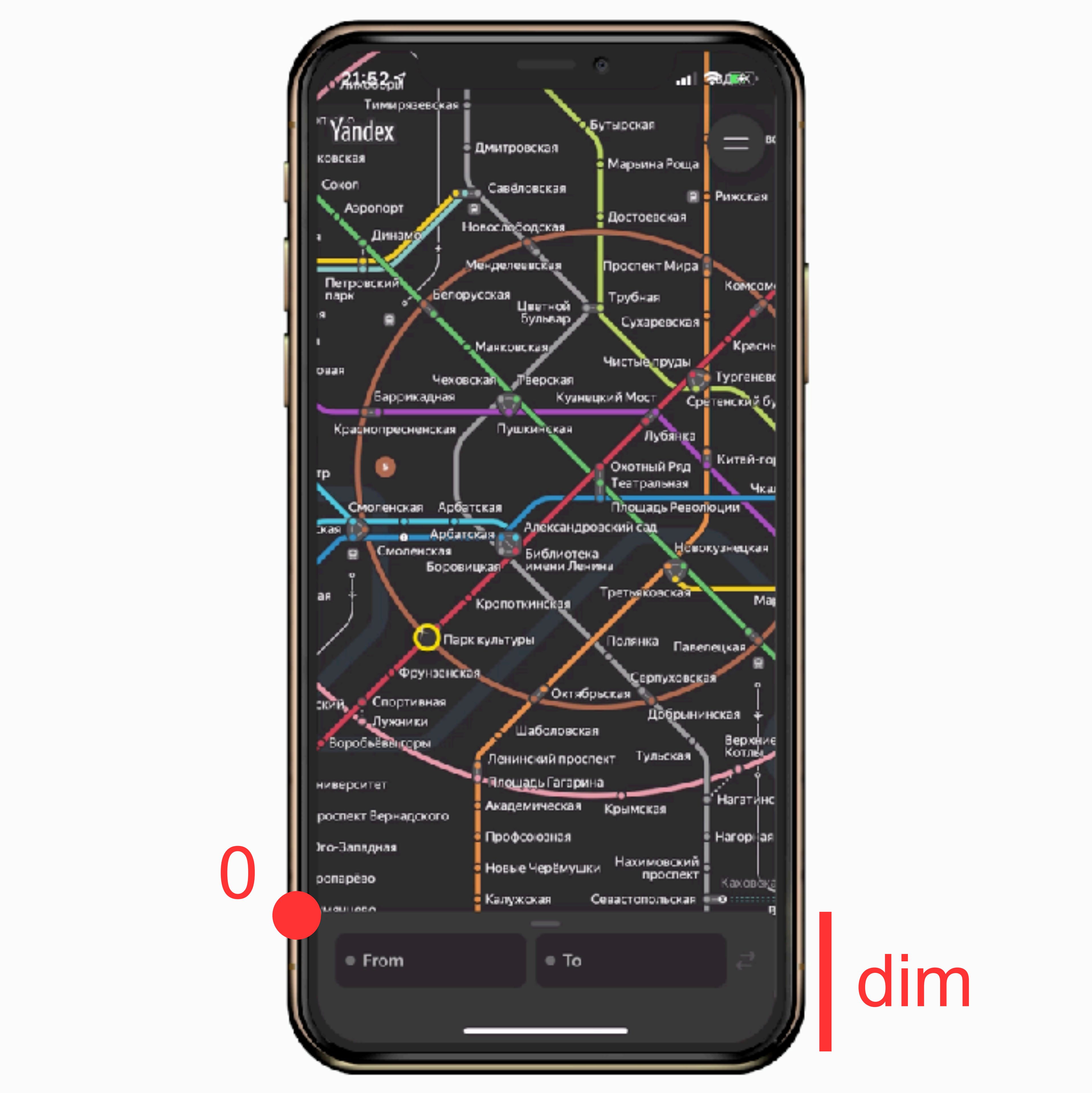
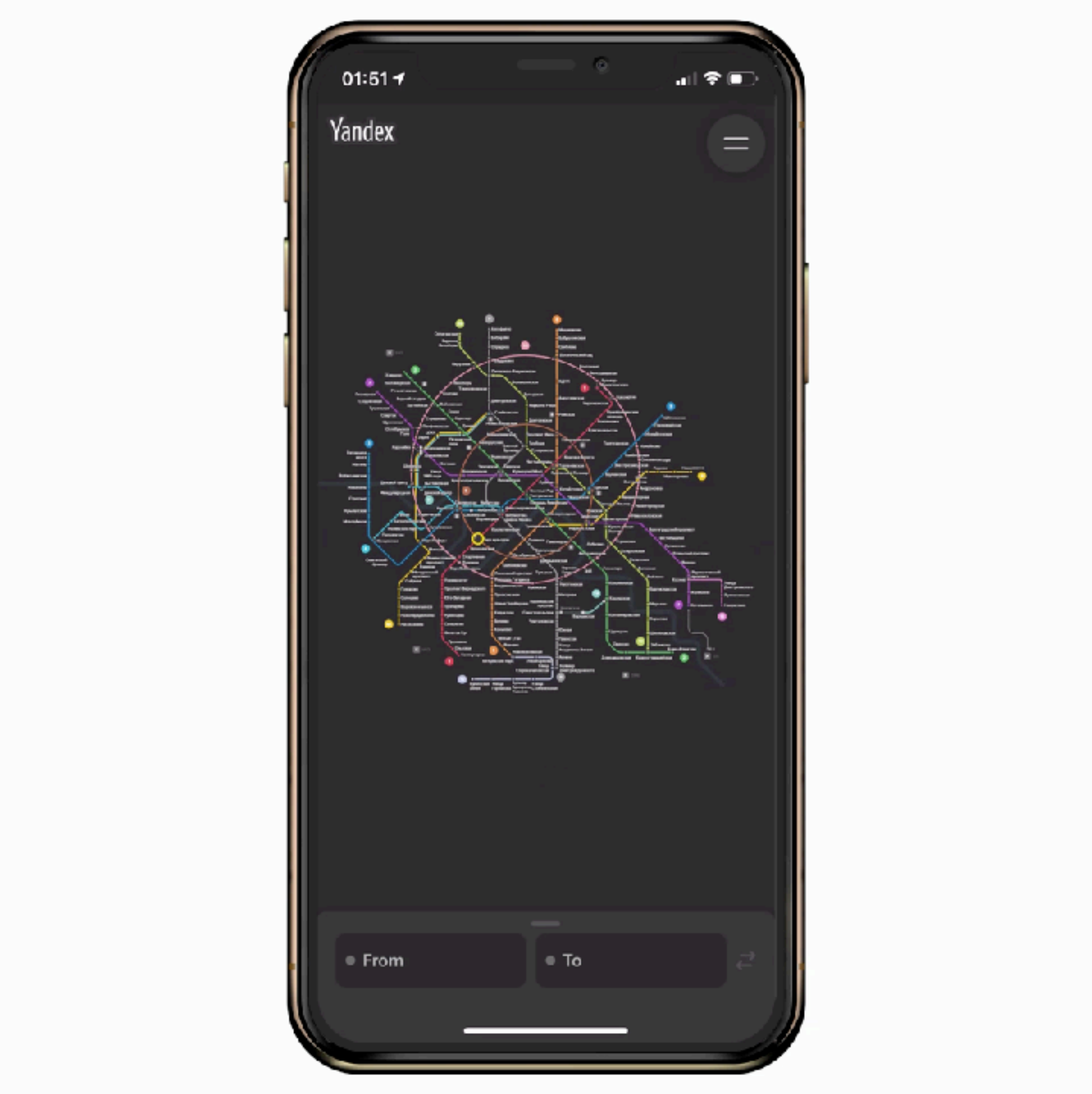


Схема Метро
Scale



Заключение

Заключение

- Плавный переход между состояниями → Проекция + Spring Animation
- Плавная граница → Rubber Band Effect

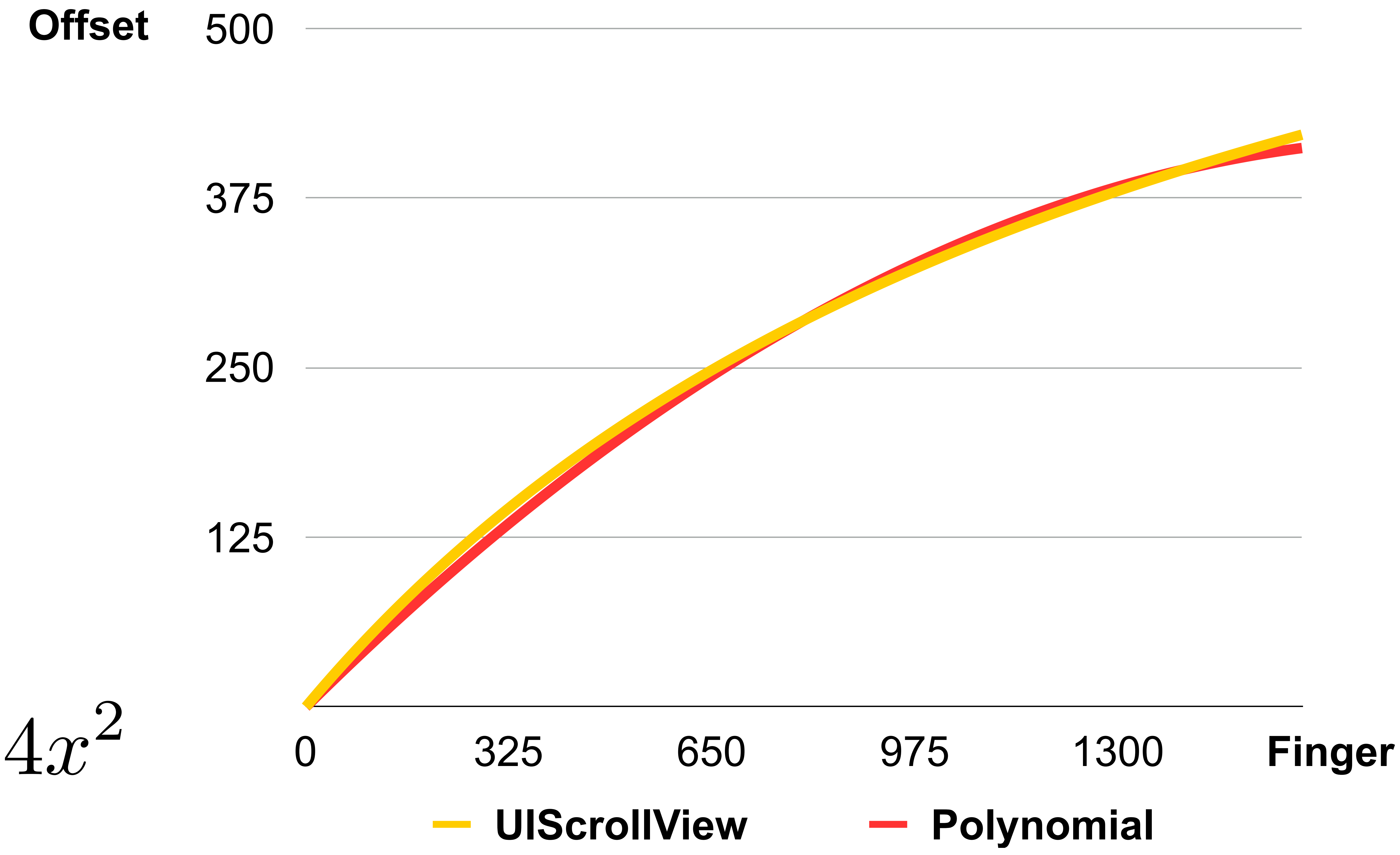
Зачем

- Rubber Band Effect

$$y = \left(1 - \frac{1}{\frac{cx}{d} + 1}\right)d$$

- Приближение

$$y = 2 + 0.456x - 0.000124x^2$$



Ссылки

- Репозиторий с примерами
  github.com/super-ultra/ScrollMechanics
- Designing Fluid Interfaces
  developer.apple.com/videos/play/wwdc2018/803
- Advanced Animations with UIKit
  developer.apple.com/videos/play/wwdc2017/230
- Блог разработки Яндекс.Карт
  medium.com/yandex-maps-ios

Яндекс

Спасибо

Илья Лобанов

ultra@yandex-team.ru

github.com/super-ultra/ScrollMechanics

medium.com/yandex-maps-ios

medium.com/esskeetit