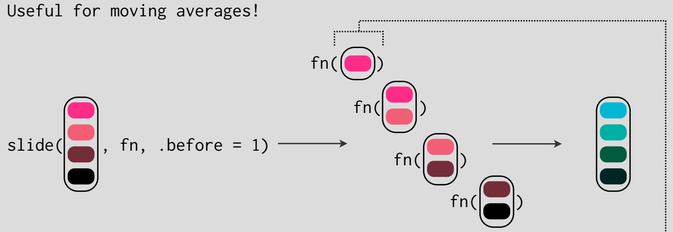
Sliding Windows and Calendars

{slider} - rstd.io/slider

Rolling Windows

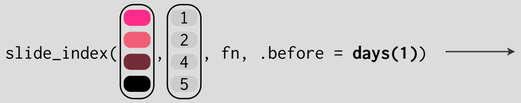
Slide variants apply a function along sequential windows of a vector.
Useful for moving averages!

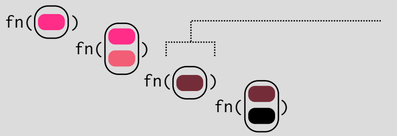


With **.complete = TRUE**, this partial window won't be evaluated.

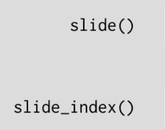
Time-Aware Sliding

Respect the "gaps" in your time series by supplying a secondary index.





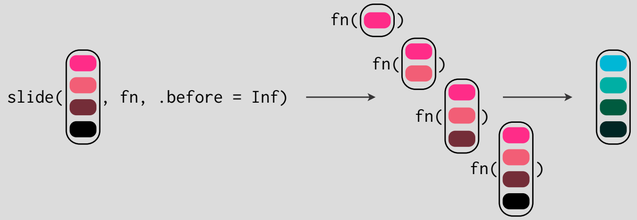
Looking back 1 day creates a range of [3, 4]. Since day 2 isn't in the range, it isn't included.



	X:	a	b	c	d	e	f	g	h	i	j
	Index:	1	2	4	5	6	7	9	10	11	13

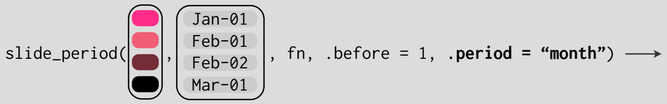
Expanding Windows

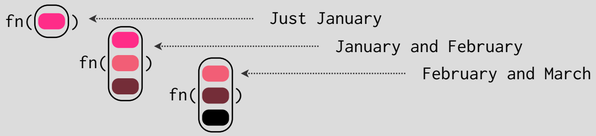
Set **.before = Inf** to use an expanding window.
Useful for cumulative functions!



Period-Blocked Sliding

Slide in rolling period chunks to summarize at different frequencies.





Familiar Syntax

If you've used purrr, you'll feel at home with slide!

Variants such as:

slide_dbl(), **slide_dfr()**

Multiple inputs with:

slide2(.x, .y), **pslide(.l)**

{almanac} - rstd.io/almanac

Recurrence Rules

Construct holiday / weekend events with recurrence rules.

```
on_labor_day <- yearly() %>%
  recur_on_ymonth("September") %>%
  recur_on_wday("Monday", nth = 1)
```

Start with a base frequency

Add recurrence conditions

Is a date **in** the recurrence set?

```
alma_in(
  c("2019-09-02", "2019-09-03"),
  on_labor_day
)
#> [1] TRUE FALSE
```

Schedules

Combine individual rules into comprehensive schedules.

```
on_weekends <- weekly() %>%
  recur_on_weekends()

sch_business <- schedule() %>%
  sch_rrule(on_labor_day) %>%
  sch_rrule(on_weekends) %>%
  sch_merge(hldy_christmas())
```

Prebuilt holidays

```
friday <- as.Date("2019-08-30")

alma_step(friday, n = 1, sch_business)
#> [1] "2019-09-03"
```

Step over the weekend and Labor Day to Tuesday

lubridate Extensions

Construct business day period objects.

```
one_day  <- days(1)
one_bday <- bdays(1, on_weekends)
```

Use them to step by irregular periods!

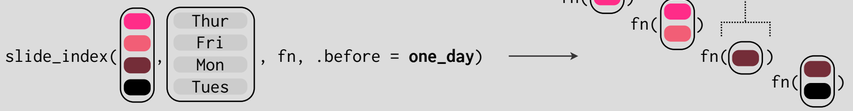
Fri + **one_day** = Sat

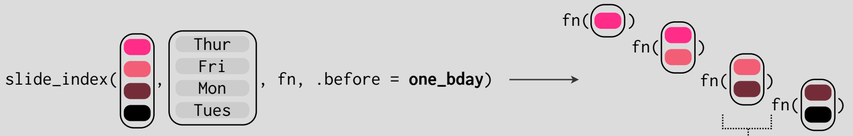
Fri + **one_bday** = Mon

{slider} + {almanac} = ❤️

Slide according to custom business schedule rules.

Looking for a range of [Sun, **Mon**], incorrect!





Correctly constructs **business** ranges of [Thur, **Fri**], [Fri, **Mon**], etc.

Davis Vaughan, @rstudio

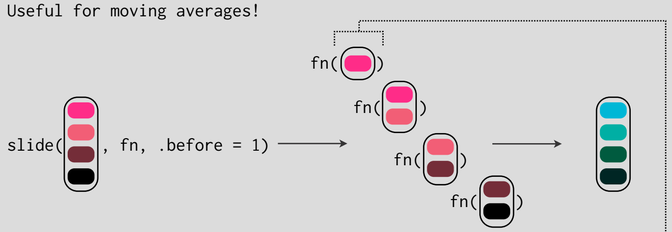
Animations use **.before = 2**.

Sliding Windows and Calendars

{slider} - r

Rolling Windows

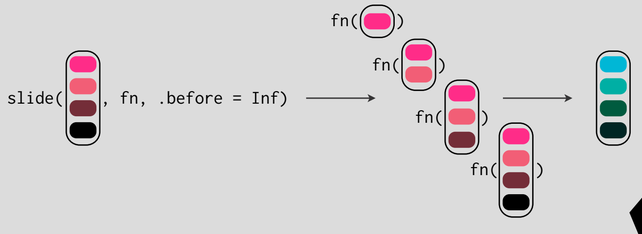
Slide variants apply a function along sequential windows of a vector.
Useful for moving averages!



With **.complete = TRUE**, this partial window won't be evaluated.

Expanding Windows

Set **.before = Inf** to use an expanding window.
Useful for cumulative functions!



Familiar Syntax

If you've used purrr, you'll feel at home with slide!

Variants such as:
slide_dbl(), **slide_dfr()**

Multiple inputs with:
slide2(.x, .y), **pslide(.l)**

liding

me series by supplying a secon

= days(1))

fn

Loo

sli

slide_index()

X:

Index:

i j

13

Peri

Slid

t Janua

Janua

01 Feb | 01 Mar | 02 May | 05 | 01 Jun | 01 Sep | 05 | 01 Oct | 01 Nov | 3 | 02 Feb

{almanac} - rstd.io/almanac

Recurrence Rules

uct holiday / weekend events with recurrence rules.

```
yearly() %>%
onth("September") %>%
wday("Monday", nth = 1)
```

Start with a base frequency

Add recurrence conditions

Is a date **in** the recurrence set?

```
alma_in(
  c("2019-09-02", "2019-09-03"),
  on_labor_day
)
#> [1] TRUE FALSE
```

Schedules

Combine individual rules into comprehensive schedules.

```
on_weekends <- weekly() %>%
  recur_on_weekends()

sch_business <- schedule() %>%
  sch_rrule(on_labor_day) %>%
  sch_rrule(on_weekends) %>%
  sch_merge(hldy_christmas())
```

Prebuilt holidays

```
friday <- as.Date("2019-08-30")

alma_step(friday, n = 1, sch_business)
#> [1] "2019-09-03"
```

Step over the weekend and Labor Day to Tuesday

lubridate Extensions

Construct business day period objects.

```
one_day  <- days(1)
one_bday <- bdays(1, on_weekends)
```

Use them to step by irregular periods!

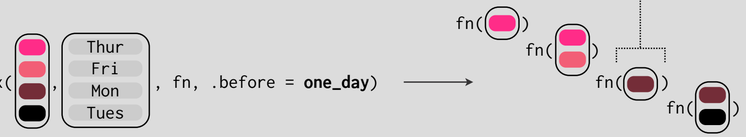
Fri + **one_day** = Sat

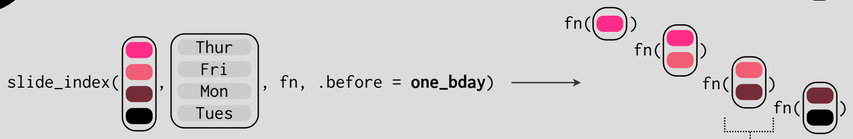
Fri + **one_bday** = Mon

{slider} + {almanac} = ❤️

Slide according to custom business schedule rules.

ooking for a range of [Sun, **Mon**], incorrect!





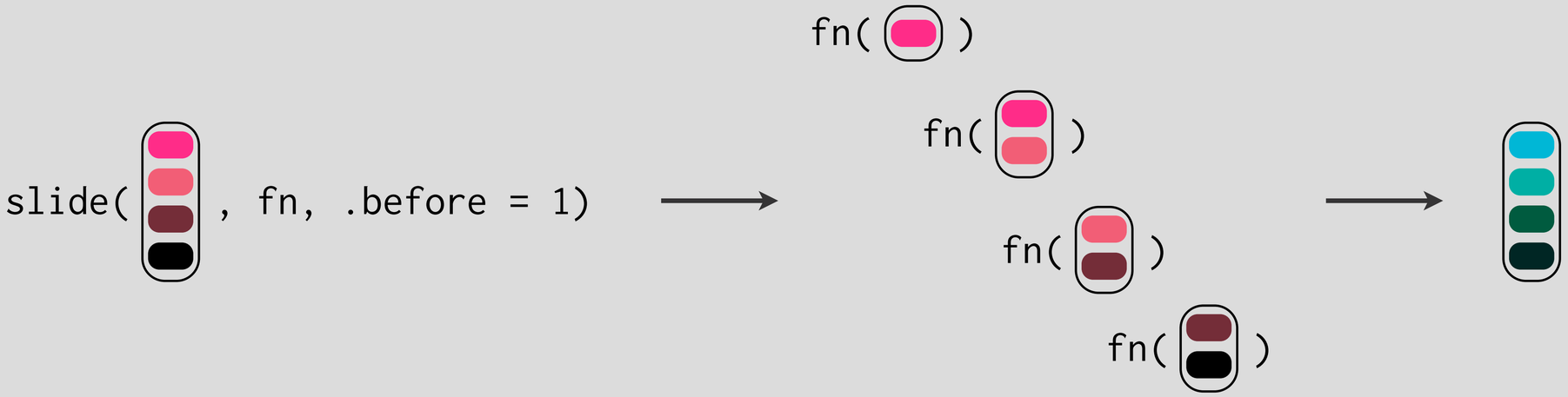
Correctly constructs **business** ranges of [Thur, **Fri**], [Fri, **Mon**], etc.

Davis Vaughan, @rstudio

Animations use **.before = 2.**

Rolling & Expanding Windows

Slide variants apply a function along sequential windows of a vector.
Useful for moving averages!



Familiar Syntax

If you've used purrr, you'll feel at home with slide!

Variants such as:

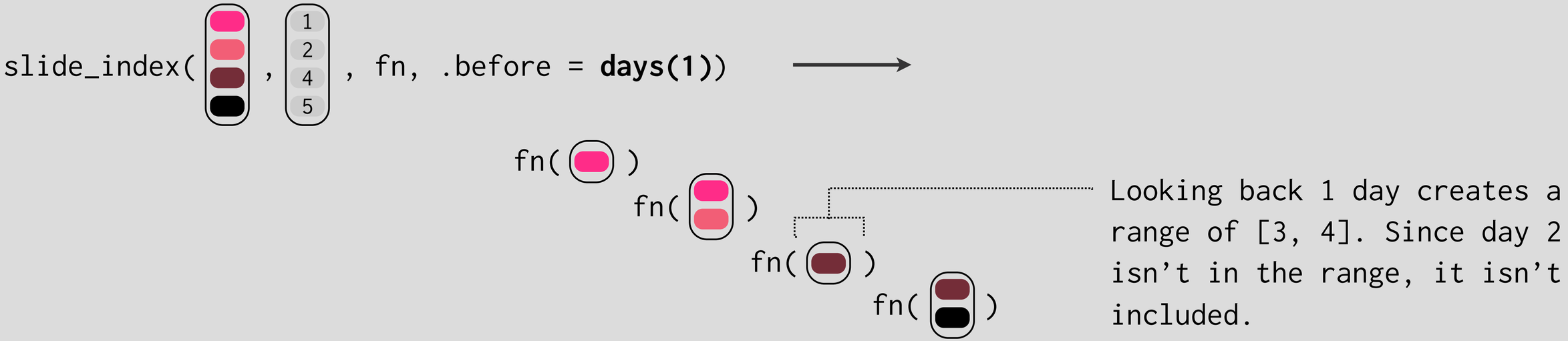
slide_dbl(), **slide_dfr()**

Multiple inputs with:

slide2(.x, .y), **pslide(.l)**

Time-Aware Sliding

Respect the “gaps” in your time series by supplying a secondary index.



Recurrence Rules

Construct holiday / weekend events with recurrence rules.

Start with a base frequency

```
on_labor_day <- yearly() %>%
  recur_on_ymonth("September") %>%
  recur_on_wday("Monday", nth = 1)
```

Add recurrence conditions

Is a date **in** the recurrence set?

```
alma_in(
  c("2019-09-02", "2019-09-03"),
  on_labor_day
)
#> [1] TRUE FALSE
```

lubridate Extensions


```
on_weekends <- weekly() %>%
  recur_on_weekends()
```

Construct business day period objects.

```
one_day  <- days(1)

one_bday <- bdays(1, on_weekends)
```

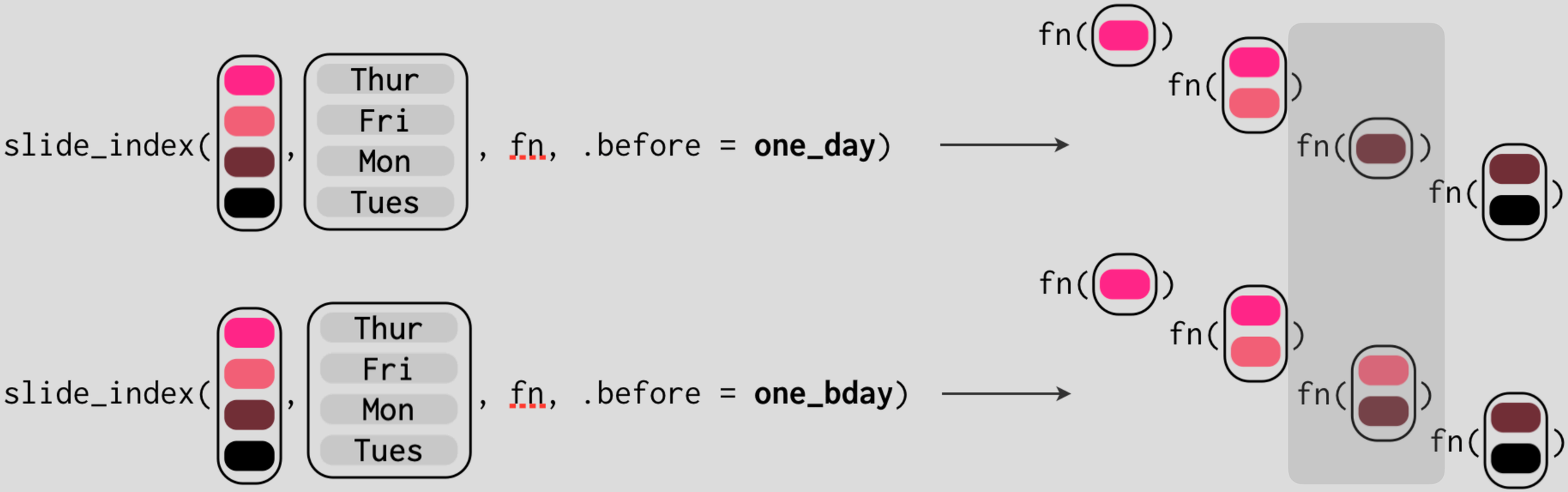
Use them to step by irregular periods!

```
Fri + one_day  = Sat

Fri + one_bday = Mon
```

{slider} + {almanac} = ❤️

Slide according to custom business schedule rules.



{slider} - rstd.io/slider

{almanac} - rstd.io/almanac

slides - github.com/DavisVaughan/rstudio-conf-2020

 @dvaughan32