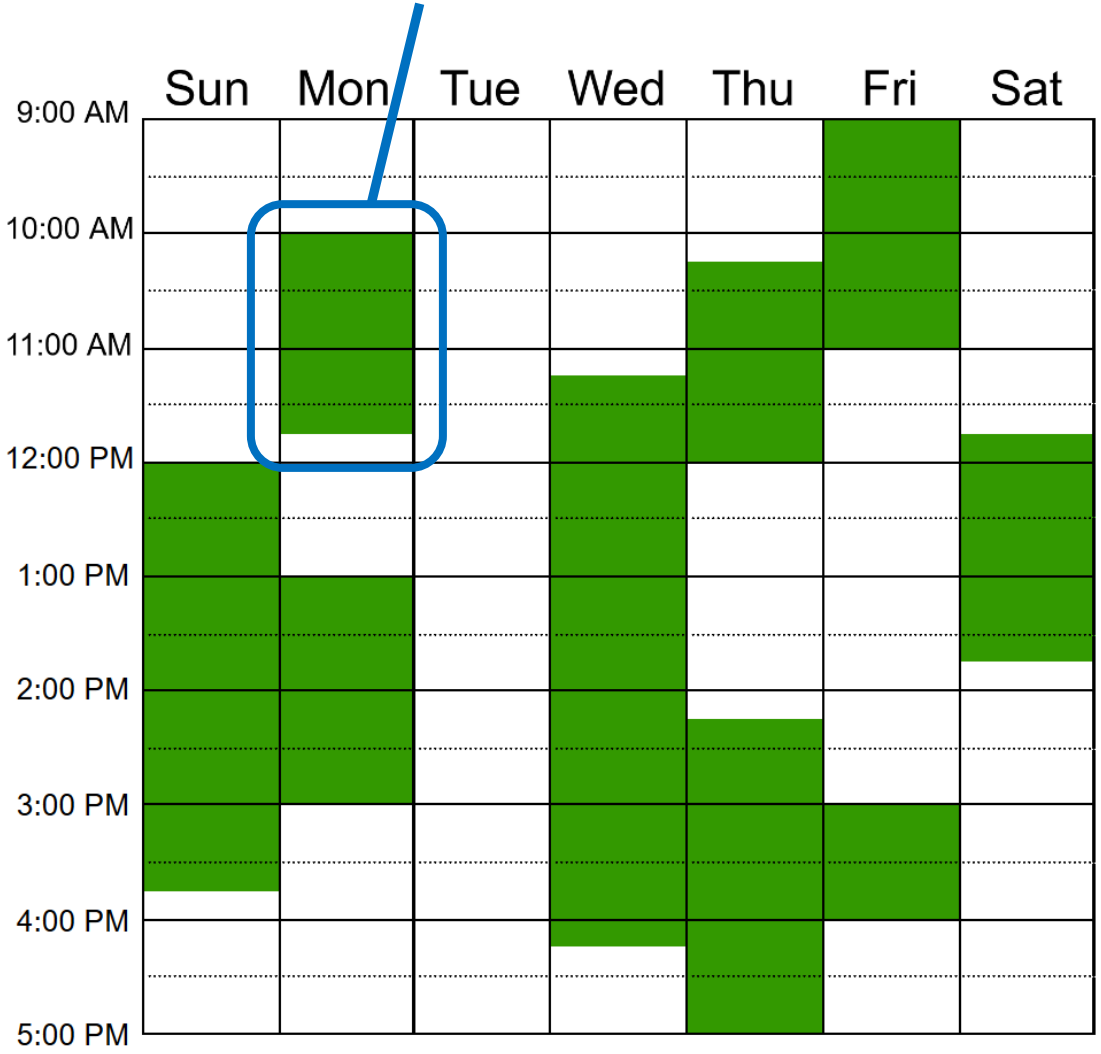
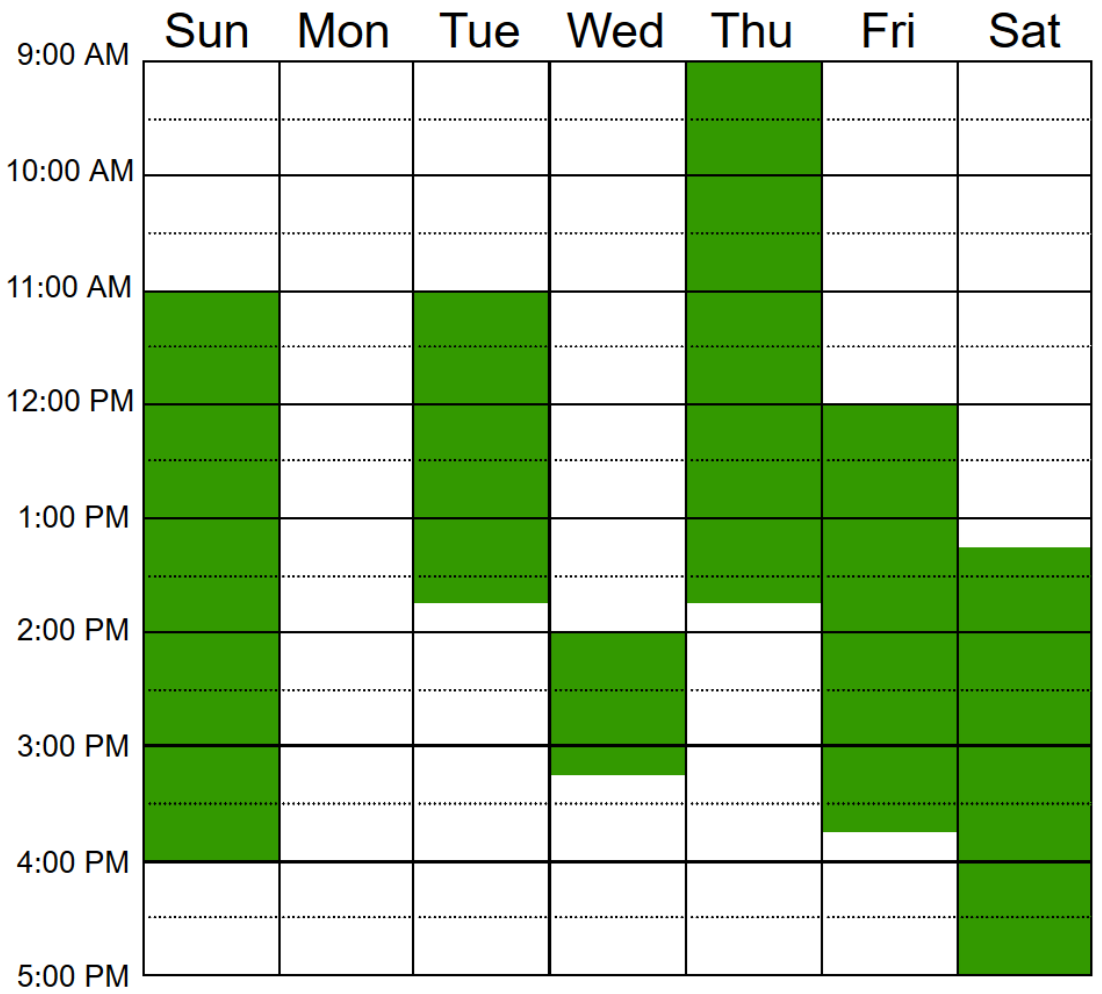
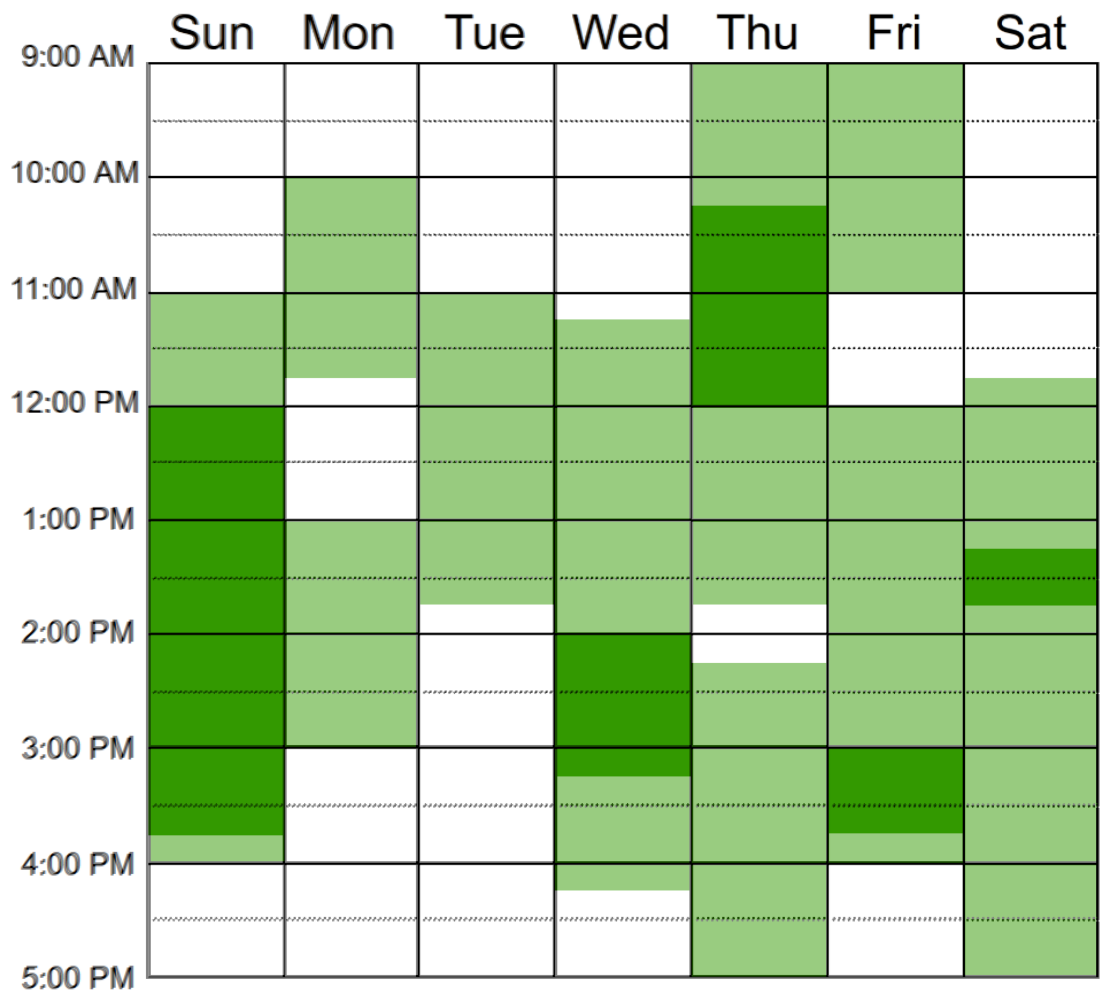
CS-320 Lab 5: when2meet

a `schedule` is a list of `intervals`

every `interval` has a day, start time, and end time






Sun
Mon
Tue
Wed
Thu
Fri
Sat
9:00 AM
10:00 AM
11:00 AM
12:00 PM
1:00 PM
2:00 PM
3:00 PM
4:00 PM
5:00 PM

`compare_time t1 t2` is > 0 iff `t1` is **later** in the week than `t2`

`is_empty i` is true iff end time is at/after start time

`compare_interval i1 i2` is > 0 iff `i1` begins **later** in day/week than `i2`

```
let is_empty (i : interval) : bool =
  compare_time i.start_time i.end_time ≥ 0

let compare_interval (i1 : interval) (i2 : interval) : int =
  let cd = compare_day i1.day i2.day in
  if cd <> 0 then cd else
  compare_time i1.start_time i2.start_time
```

prioritize whether days differ

if same day, check start time

`intersect_i_i i1 i2` is the interval overlap of `i1` and `i2`

overlap can only exist if both are on the same day. Consider using `min` and `max`

`intersect_s_i s i` is the overlap of `i` with all intervals in `s`

consider using `List.filter`, `List.map`, and/or `List.filter_map`

`intersect_s_s s1 s2` is the overlap all intervals in `s1` with all in `s2`

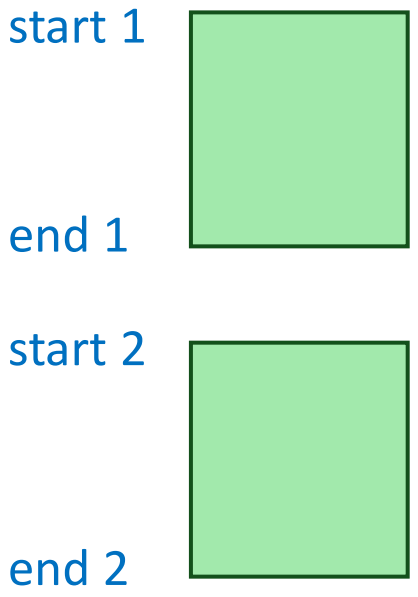
use `List.fold_left` ~~`or List.fold_right`~~

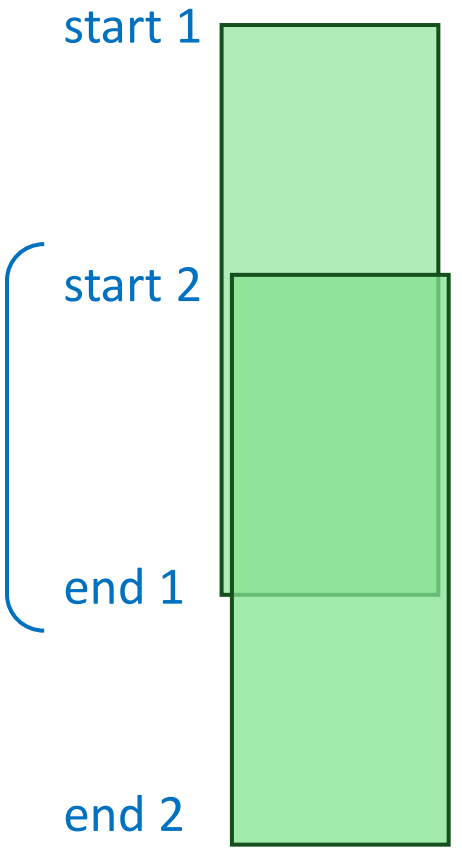
`intersect_schedules [s1; ... ;sn]` is the overlap all intervals in all s_i

use `List.fold_left` ~~`or List.fold_right`~~

INTERVAL / INTERVAL

```
let intersect_i_i i1 i2 : interval option =
  if i1.day <> i2.day then None else
  let day = i1.day in
  let start_time =
      max compare_time i1.start_time i2.start_time in
  let end_time =
      min compare_time i1.end_time i2.end_time in
  let i = {day;start_time;end_time} in
  if is_empty i then None else Some i
```





SCHEDULE / INTERVAL

```
let intersect_s_i (s : schedule) (i: interval) : schedule =
  List.filter_map (intersect_i_i i) s
```

map over `s`, intersecting each interval in `s` with `i`

filter out all *none* results

SCHEDULE / SCHEDULE

```
let intersect_s_s (s1 : schedule) (s2: schedule) : schedule =
  List.fold_left (fun acc i → (intersect_s_i s1 i) @ acc) [] s2
```

initially empty accumulator

fold over each interval in s2

accumulate intersections of s1 with each interval in s2

SCHEDULE / ⋯ / SCHEDULE

```
let identity_schedule : schedule =
  let all_day d = {
    day = d;
    start_time = {hour=0; minute=M00};
    end_time   = {hour=24;minute=M00};
  } in
  List.map all_day [Mo;Tu;We;Th;Fr;Sa;Su]
```

schedule containing
all days and times

```
let intersect_schedules (ss : schedule list) : schedule =
  List.fold_left intersect_s_s identity_schedule ss
```

start with identity schedule, then fold over each
schedule by chipping away at accumulator each time
until intersection of all schedules remains