6.205
(aka 6.111)

Sequential Logic III

Video, Pipelining

Interfacing to Sequential Logic

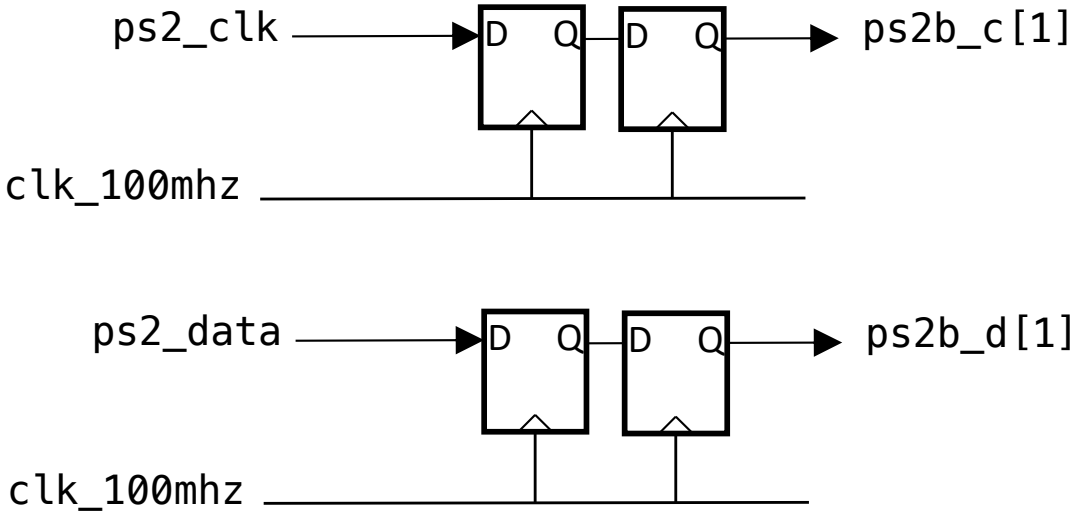
…Or what are the problems with working with Sequential Logic?....

Huh?

- In Lab 2:

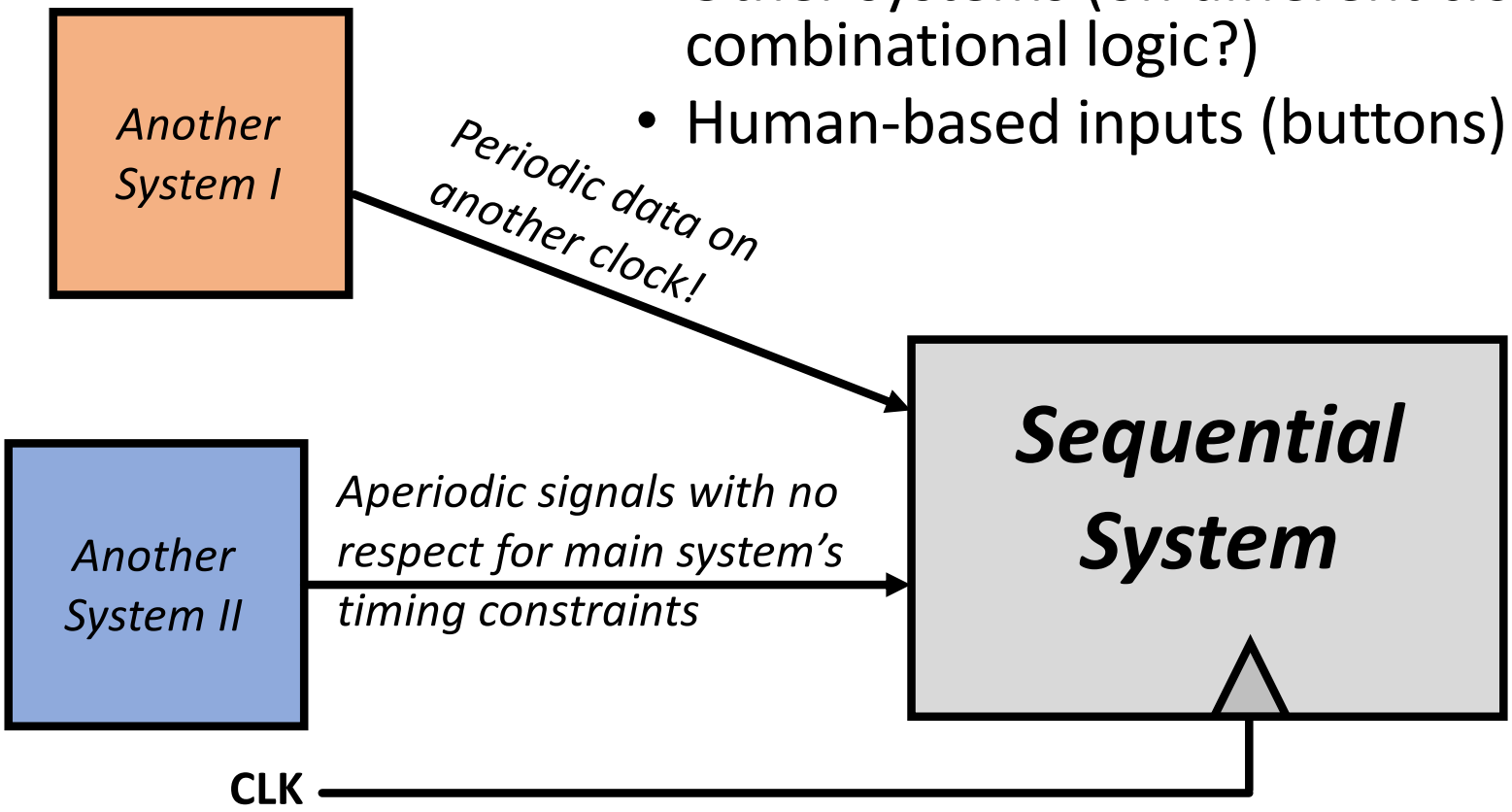
```
logic [1:0] ps2b_c;
logic [1:0] ps2b_d;

always_ff @(posedge clk_100mhz)begin
  ps2b_c[0] <= ps2_clk;
  ps2b_d[0] <= ps2_data;
  ps2b_c[1] <= ps2b_c[0];
  ps2b_d[1] <= ps2b_d[0];
end
```

- Basically builds this:



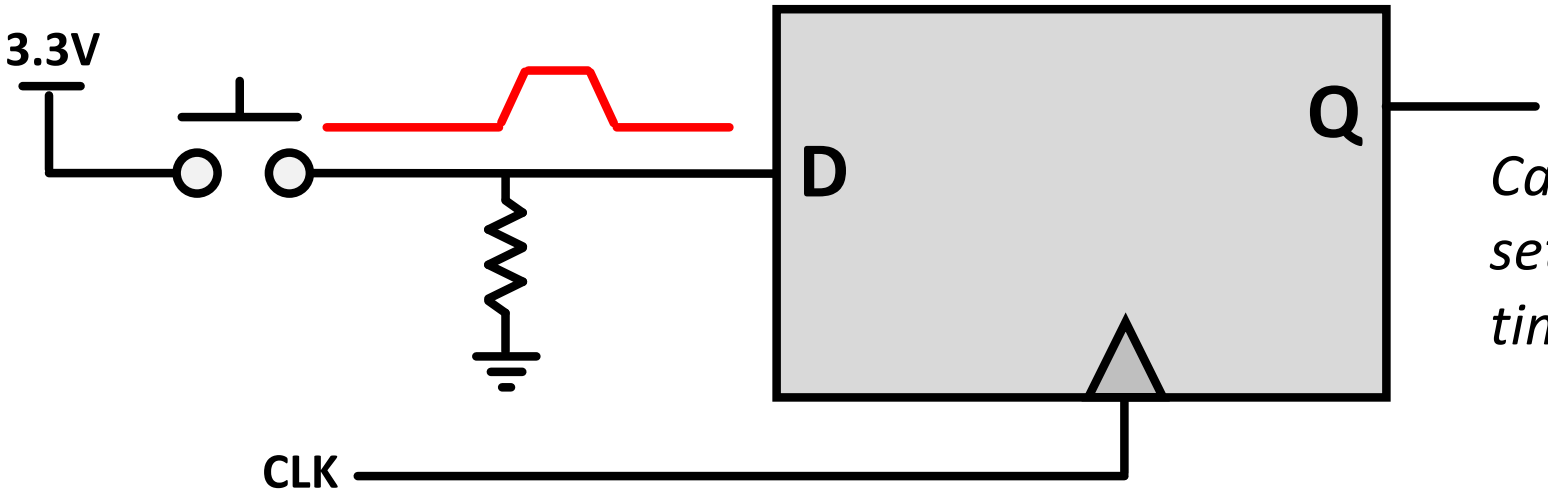
What if…?

- …we need to interface with outside equipment:
  - Other systems (on different clocks or from combinational logic?)
  - Human-based inputs (buttons)



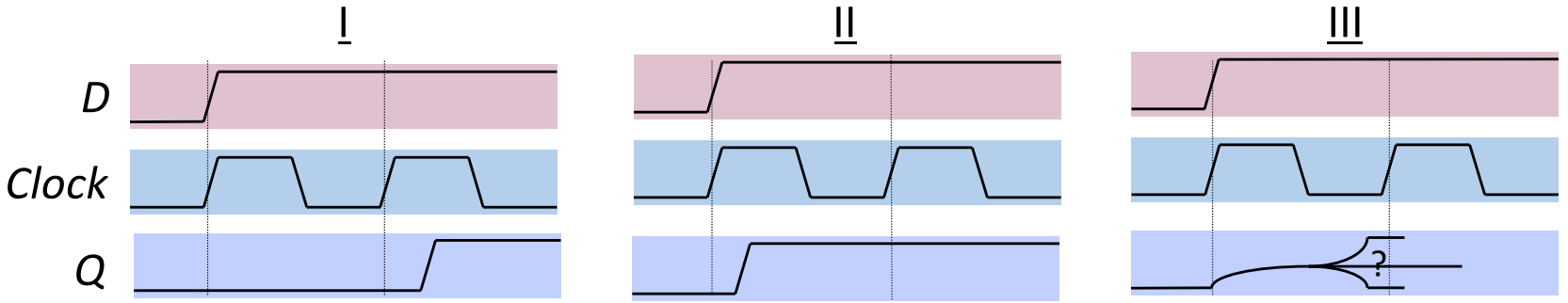
Can't guarantee setup and hold times will be met!

Example: Asynchronous Inputs in Sequential Systems



Can't guarantee setup and hold times will be met!

When an asynchronous signal causes a setup/hold violation...



Transition is missed on first clock cycle, but caught on next clock cycle.

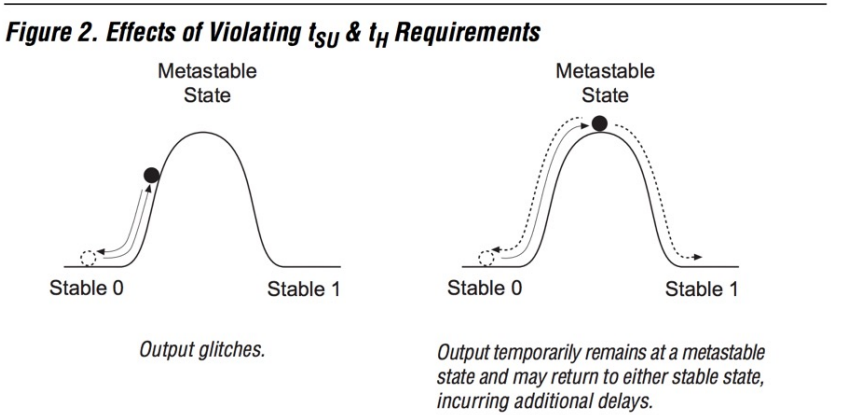
Transition is caught on first clock cycle.

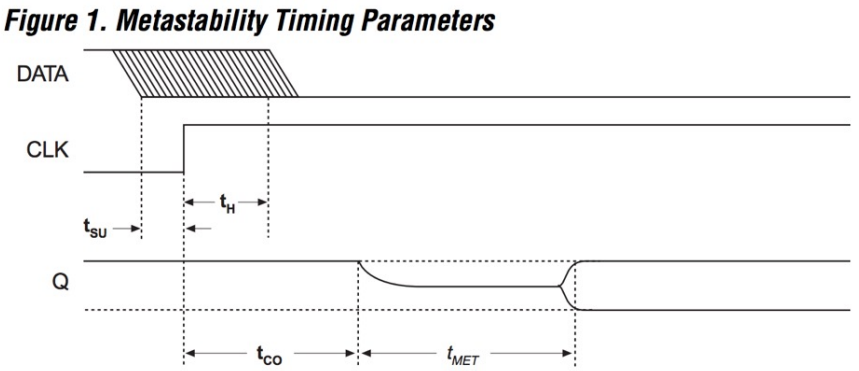
Output is metastable for an indeterminate amount of time.

Q: Which cases are problematic?

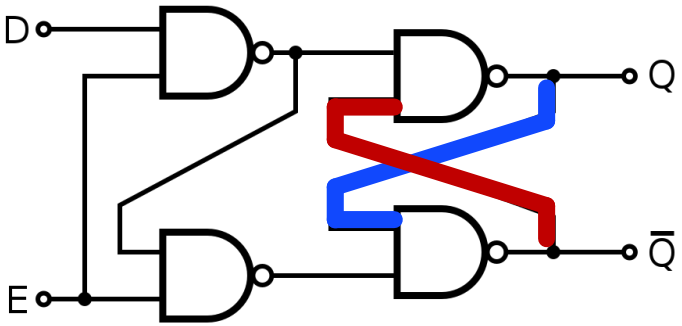
Metastability

- D-registers have issues with all that feedback and stuff going on. Can go **metastable**
- Metastability is where the system hovers between Logic High and Logic Low in an unpredictable way





Metastability in Altera (☹) Devices
Altera Application Note 42 (1999)



t_{co} = "min time from clock to output"
….think of it as t_{pd} here (not exactly the same,

Handling Metastability

- Can't globally prevent metastability, but can isolate it!
- Stringing several registers together can isolate any freakouts!

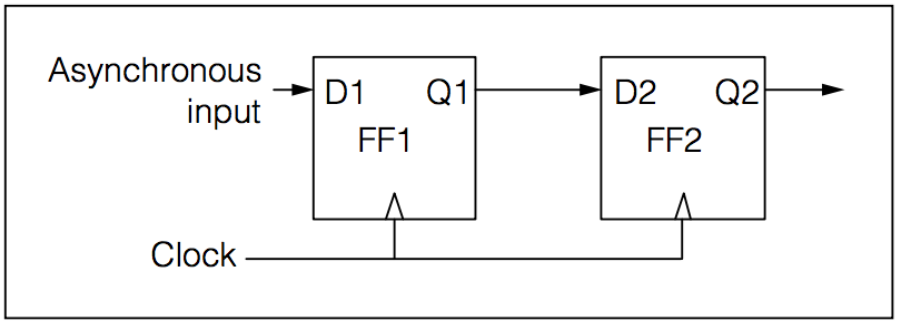
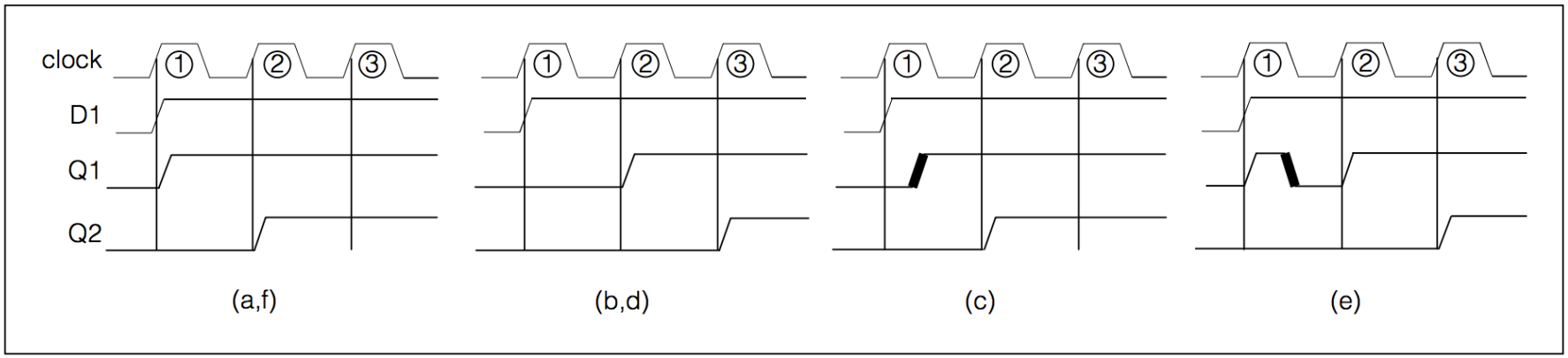


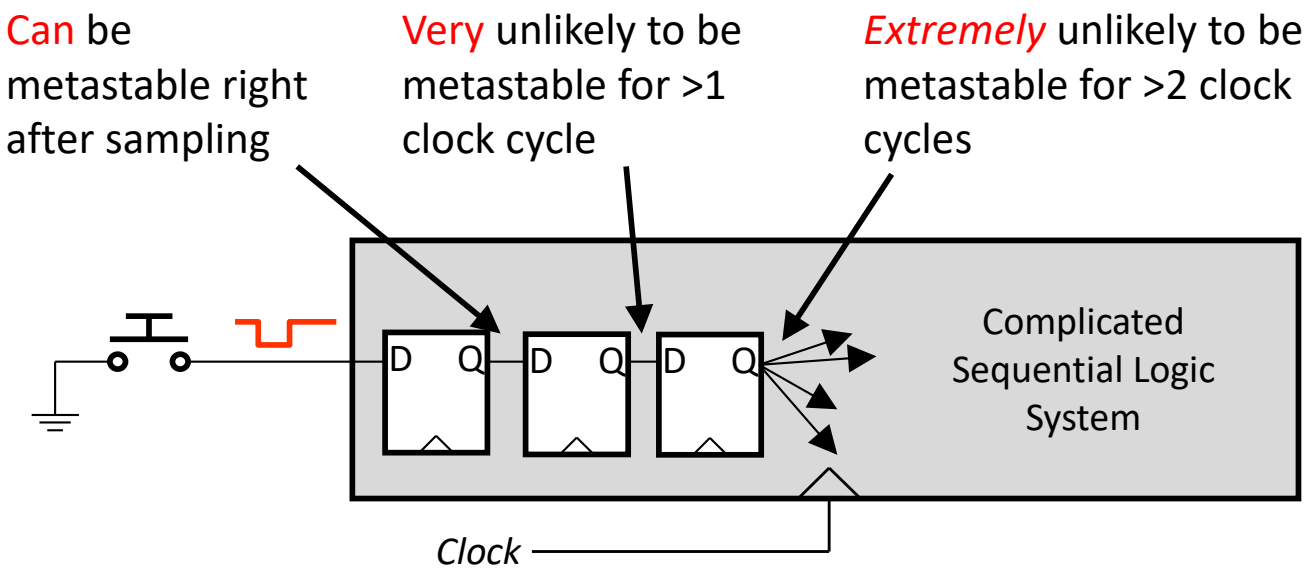
Figure 8. Two-flip-flop synchronization circuit.



"Metastability and Synchronizers: A Tutorial"
Ran Ginosar, Technion Israel Institute of Technology

Handling Metastability

- Completely preventing metastability turns out to be an impossible problem
- High gain of digital devices makes it likely that metastable conditions will resolve themselves quickly
- Solution to metastability: allow time for signals to stabilize

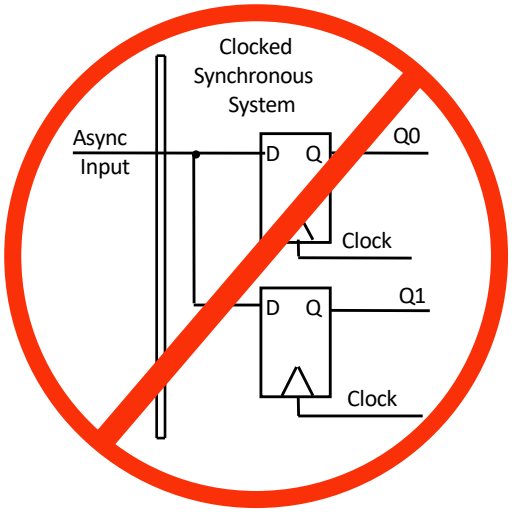


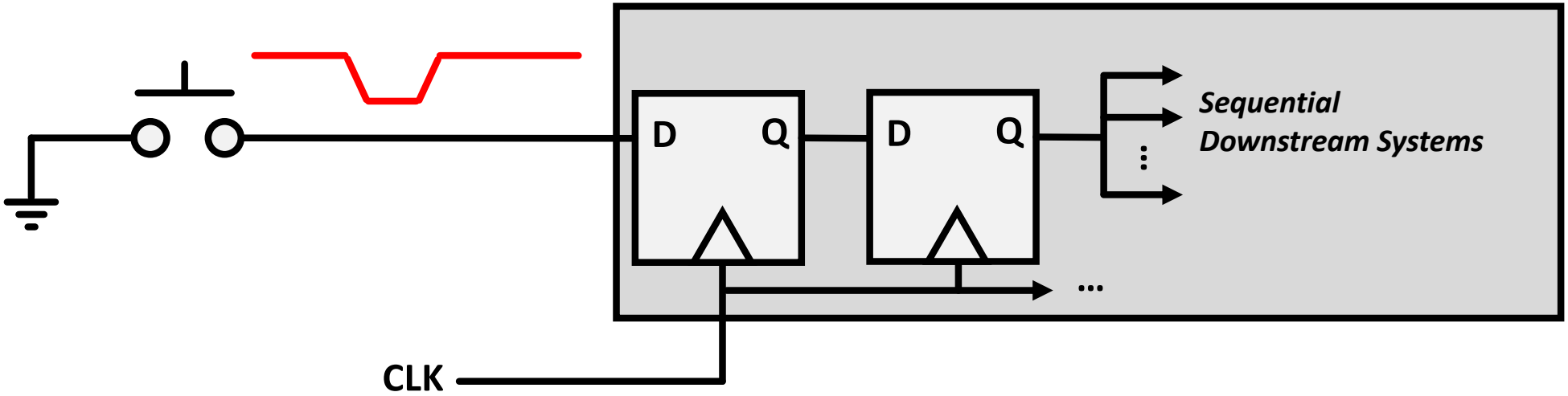
How many registers are necessary in 6.205?

- Depends on many design parameters (clock speed, device speeds, …)
- In 6.205, **a pair of synchronization** registers is sufficient
- And for simple designs…with low t_{pd} you may not even need anything

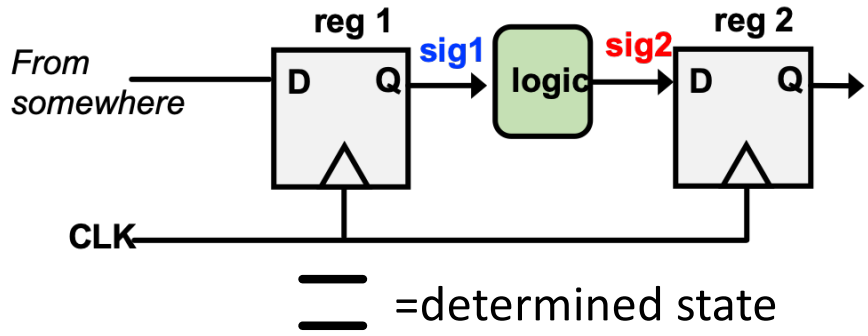
Handling Metastability

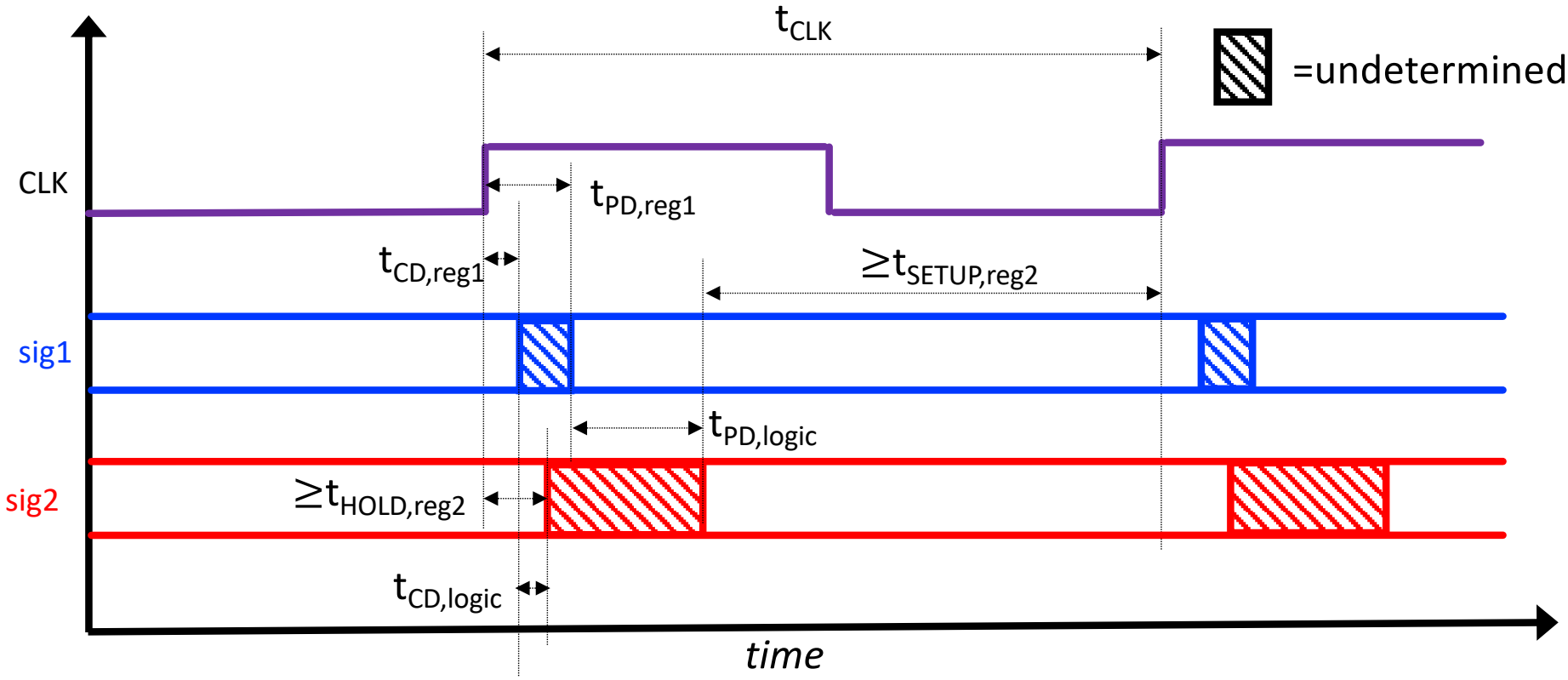
- Don’t break off an asynchronous input until it has gone through some registers
- Basically: Ensure that external signals feed **exactly one** flip-flop chain before branching





D Register Timing 2



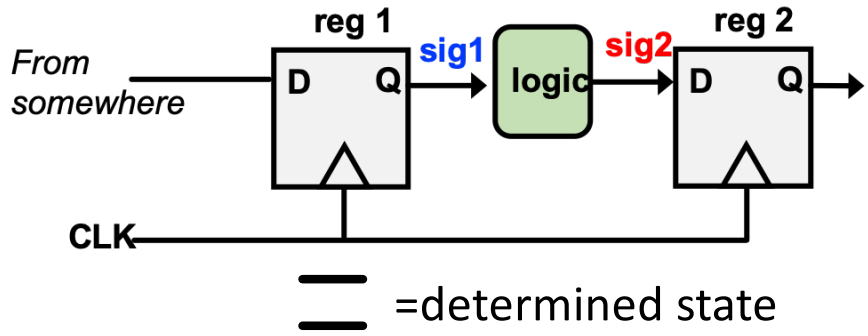


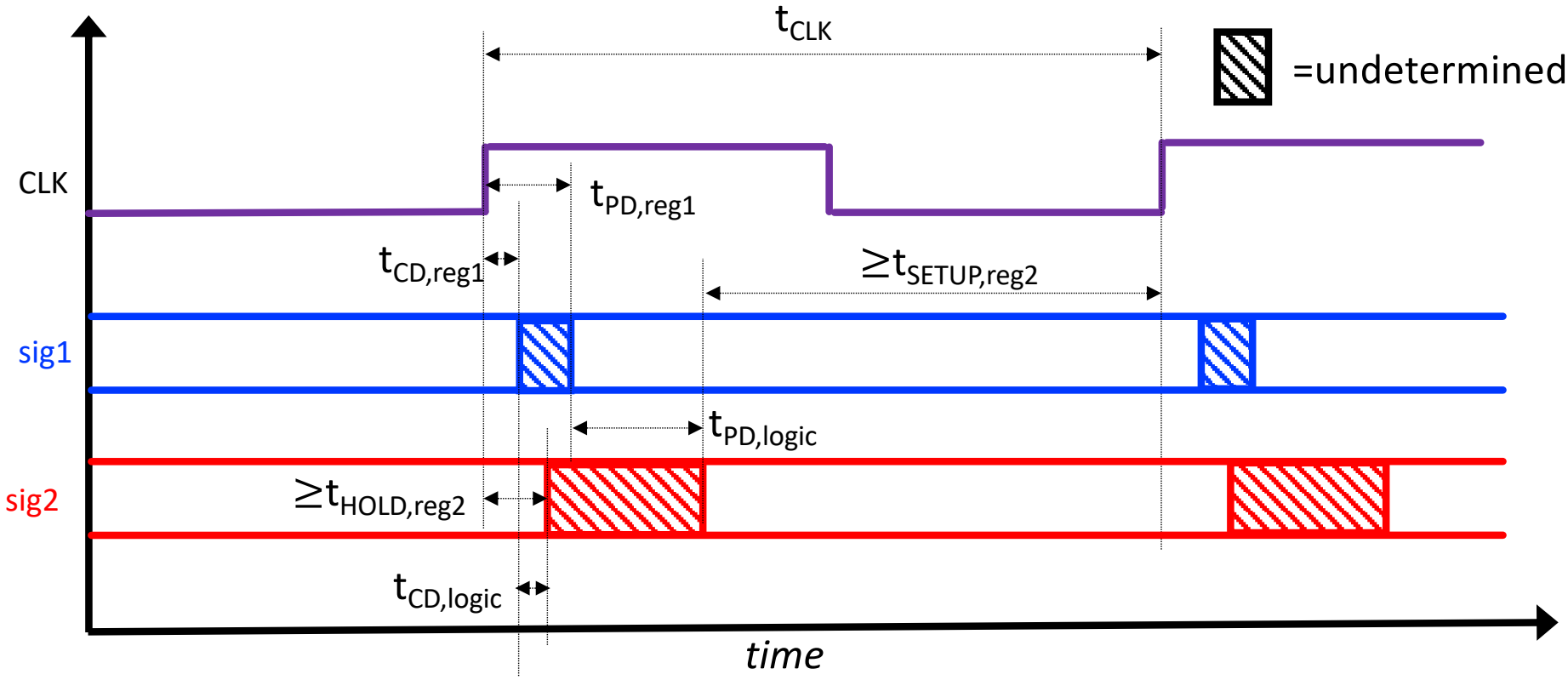
Two Requirements/ Conclusions:

$$t_{PD,reg1} + t_{PD,logic} + t_{SETUP,reg2} \leq t_{CLK}$$

$$t_{CD,reg1} + t_{CD,logic} \geq t_{HOLD,reg2}$$

D Register Timing 2





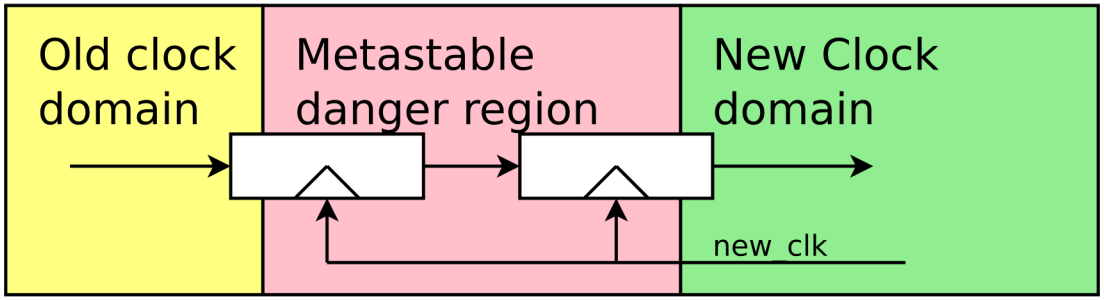
Two Requirements/ Conclusions:

$$t_{PD,reg1} + t_{met} + t_{PD,logic} + t_{SETUP,reg2} \leq t_{CLK}$$

$$t_{CD,reg1} + t_{CD,logic} \geq t_{HOLD,reg2}$$

Clock Domain Crossing

- For example:
  - Data gets sent in at 25 MHz from one device (running on its own clock)
  - Your system runs at 50 MHz




```
1  //xfer_pipe can be >2 bits wide (2 is usually fine...3 better)
2  always_ff @(posedge new_clock)
3    { new_val, xfer_pipe } <= { xfer_pipe, i_val };
```

- This only works when original clock domain frequency is less than or equal to new clock domain frequency

https://zipcpu.com/blog/2017/10/20/cdc.html

Asychronous vs. Synchronous Resets

- You will sometimes see this in HDL floating around:
  - Called “Asynchronous reset”: Reset can happen at any time…NOT ONLY on rising edge of clock

```
always_ff @(posedge clk_in, posedge rst_in)begin
  if(rst_in)begin
    //reset stuff
  end else begin
    //regular stuff
  end
end
```

- For our class just do “synchronous” resets:

```
always_ff @(posedge clk_in)begin
  if(rst_in)begin
    //reset stuff
  end else begin
    //regular stuff
  end
end
```

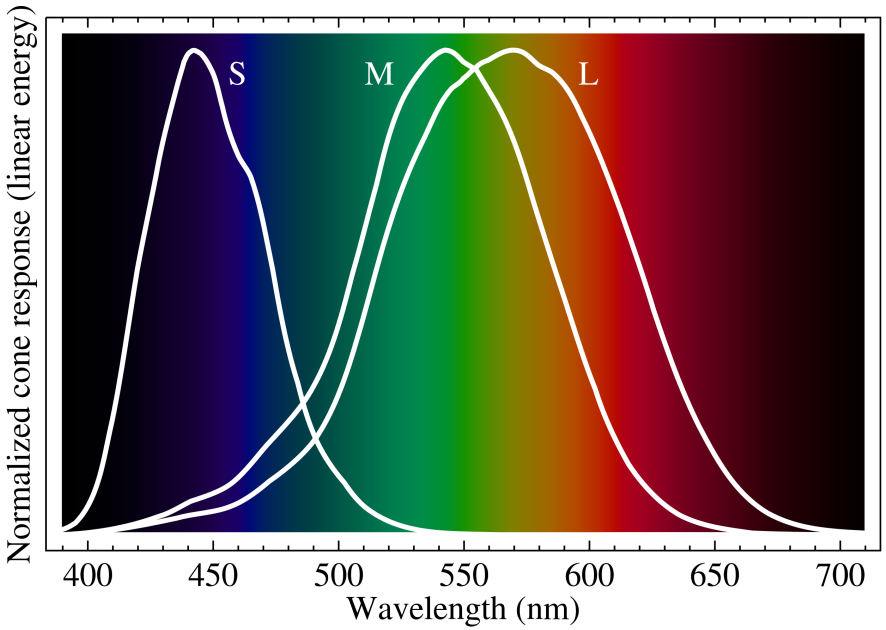
- Works better with Xilinx FPGA architectures.
- Asyncn rst can be finicky in high-speed systems

https://docs.xilinx.com/r/en-US/ug949-vivado-design-methodology/Synchronous-Reset-vs.-Asynchronous-Reset

Video

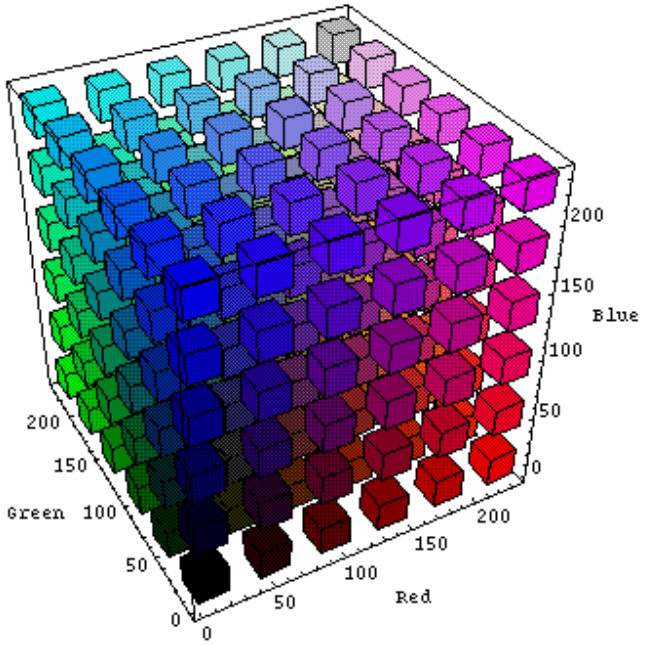
Displays are for Eyes

- Human color perception comes from three types of cone cells in the center of the eye. Each type generally has an abundance of one photoreceptive protein (which causes electrical stimulation):
  - S cones with protein from **OPN2** gene
  - M cones with protein from **OPN1MW** gene
  - L cones with protein from **OPN1LW** gene
- A human eye therefore has three independent inputs regarding visual EM radiation
- Called **”trichromatic”**



Color Space

- Human trichromatic vision is comprised of three inputs, therefore the most general way to describe these inputs is in a 3-dimensional space
- Because the L, M, and S cones “roughly” line up with Red, Green, and Blue, respectively a RGB space is often the most natural to us
- There are others, though



One form of RGB space (not the only way to display it)

https://engineering.purdue.edu/~abe305/HTMLS/rgbspace.htm

Worst Case Scenario

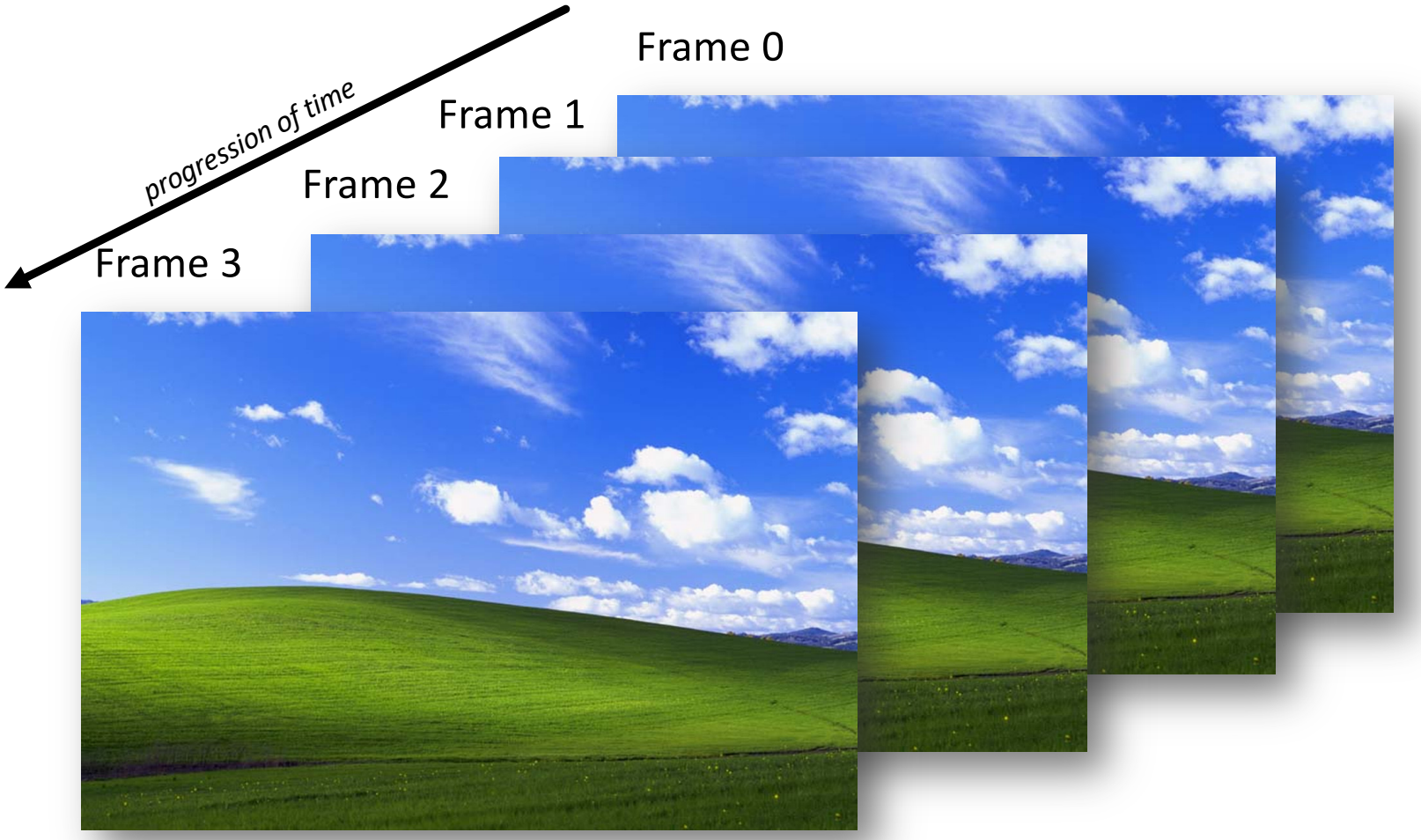
- If a person has all color receptors working…
- because of noise limitations in our naturally-evolved encoding scheme that communicates from the cone cells up to the brain…
- we can perceive about 7-10 million unique colors depending on your research source…
- How many bits do we need to encode all possible colors for this worst case?

Image or Frame

- An image/frame can be thought of as a 2-dimensional array of 3-tuples:
  - 2 spatial dimensions
  - 3 color dimensions
- Each color tuple is a "pixel"

Video (just draw a bunch of frames quickly)

- Rely on the poor RC time constants of our eye's to "fake" motion.



How to Transmit 5/6-dimensional data?

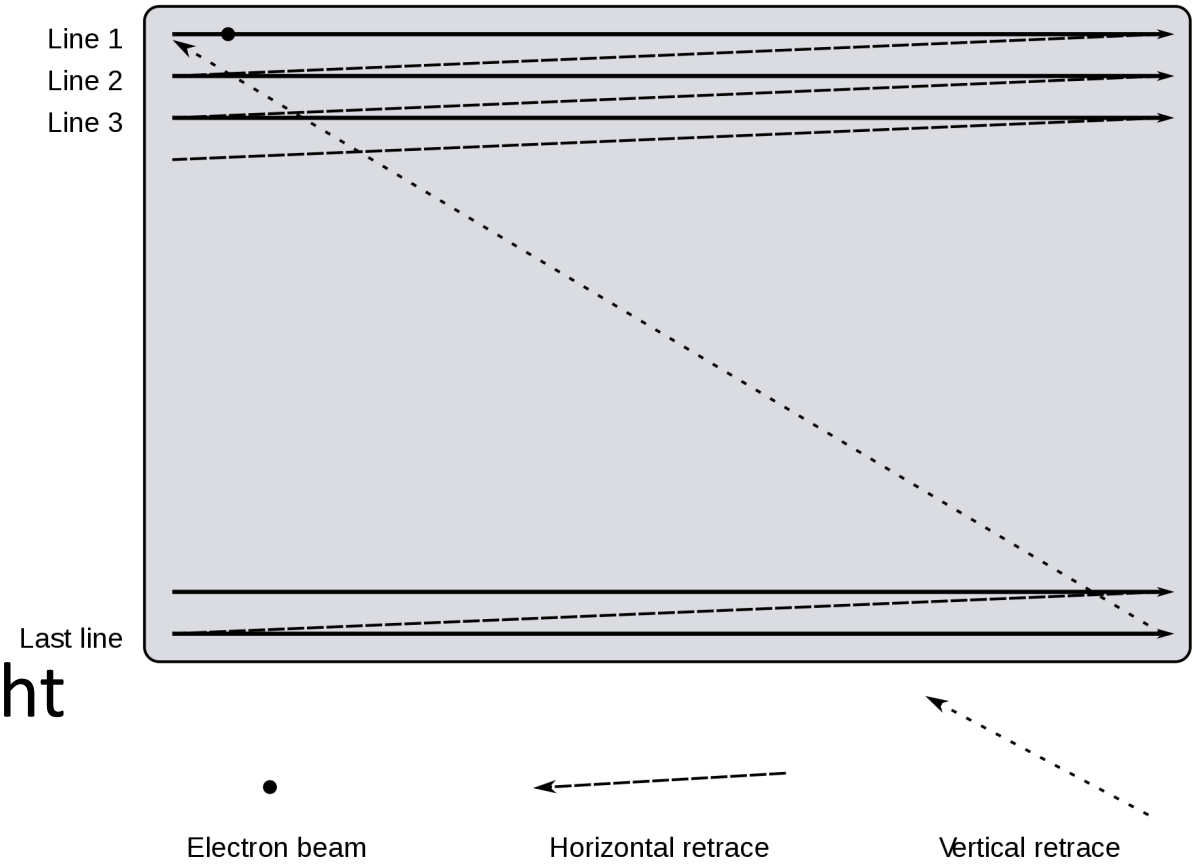
- Ideally need to convey enough 5D values quickly enough to render images fast enough that they show up as one…
- AND we also need to do the above fast enough so that fresh images appear quickly enough

How to Draw: The Raster Scan

- Spread the drawing out over time
- Images are drawn on a display almost invariably in a “raster” pattern.
- The sequence starts in the upper left, and pixels are drawn:

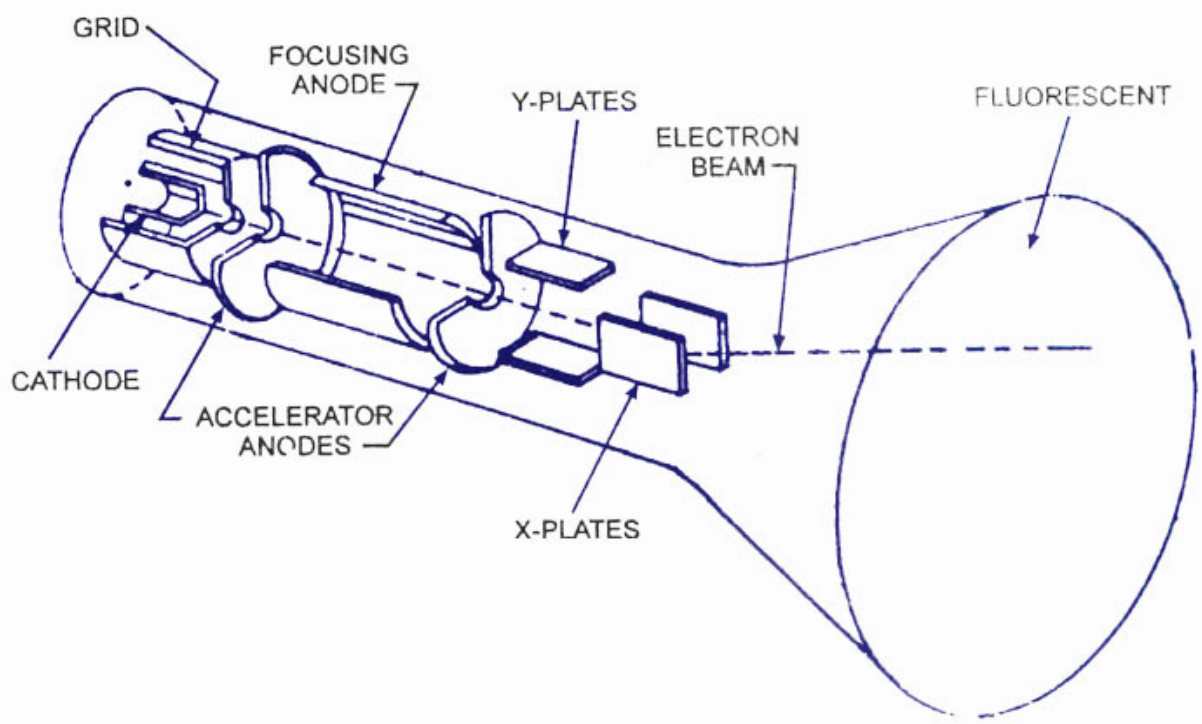
Rastrum, used for drawing musical staff

- Left→ Right
- Down a line/back
- Left→ Right
- Down a line/back
- Etc…
- End at bottom right
- Return to top left



Raster Scan Became Norm because of Early Tech (Cathode Ray Tube)

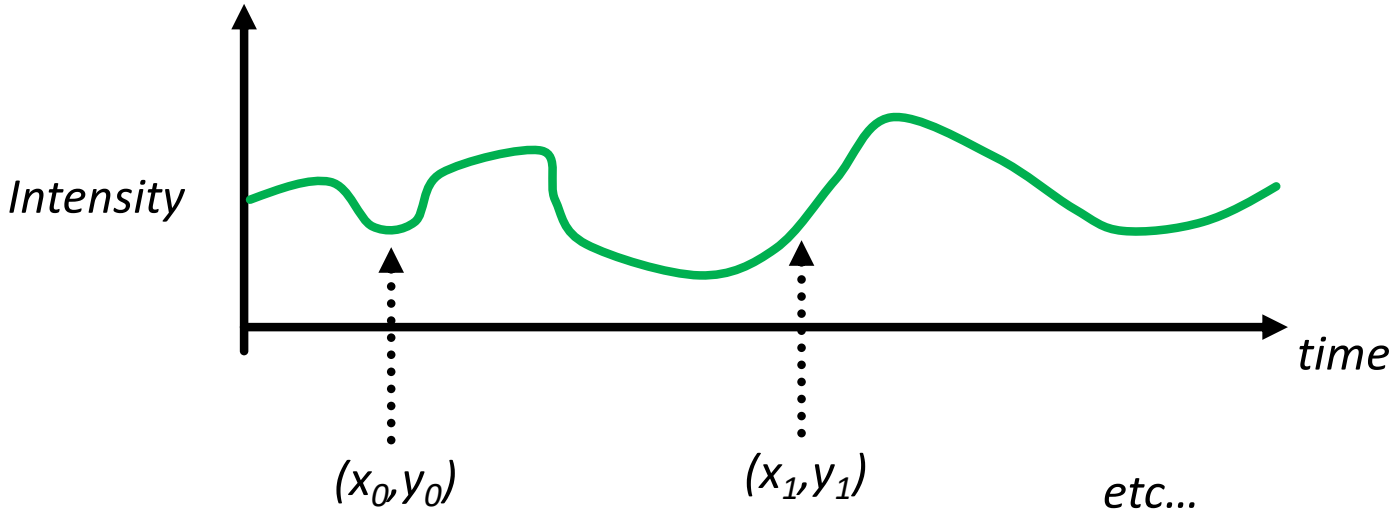
- Electron beam of varying intensity would be quickly rastered on a fluorescent screen making image



Cathode Ray Tube

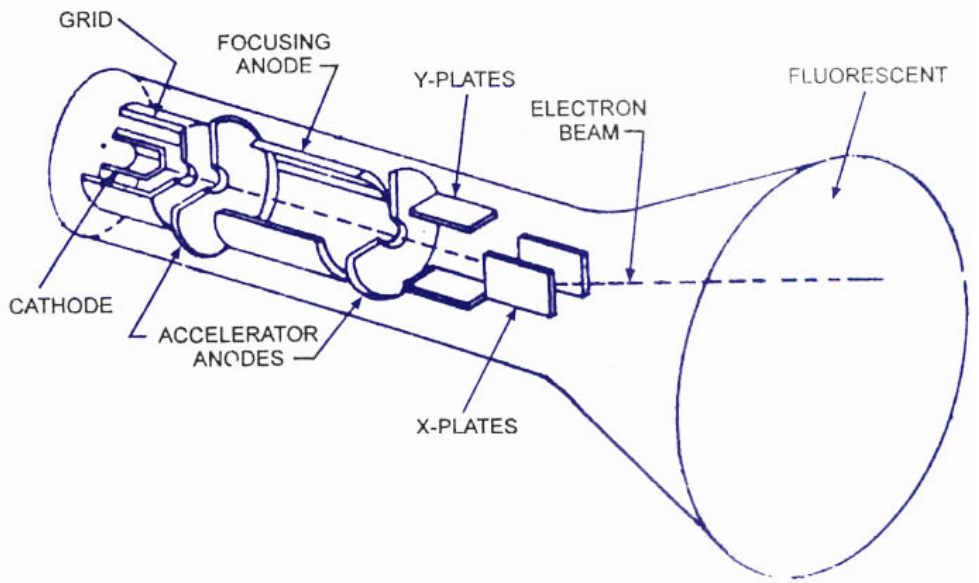
Raster Pattern of Drawing

- Allows time ←→position!
  - Takes care of two of the dimensions of info we need to convey!



First Video (Black and White)

- Early technologies prevented ability to detect and display color.
- Instead only **brightness (Luminance)** of the image was transmitted/rendered since color couldn't be rendered anyways
- So transmitting an image only involved 3 dimensions of information

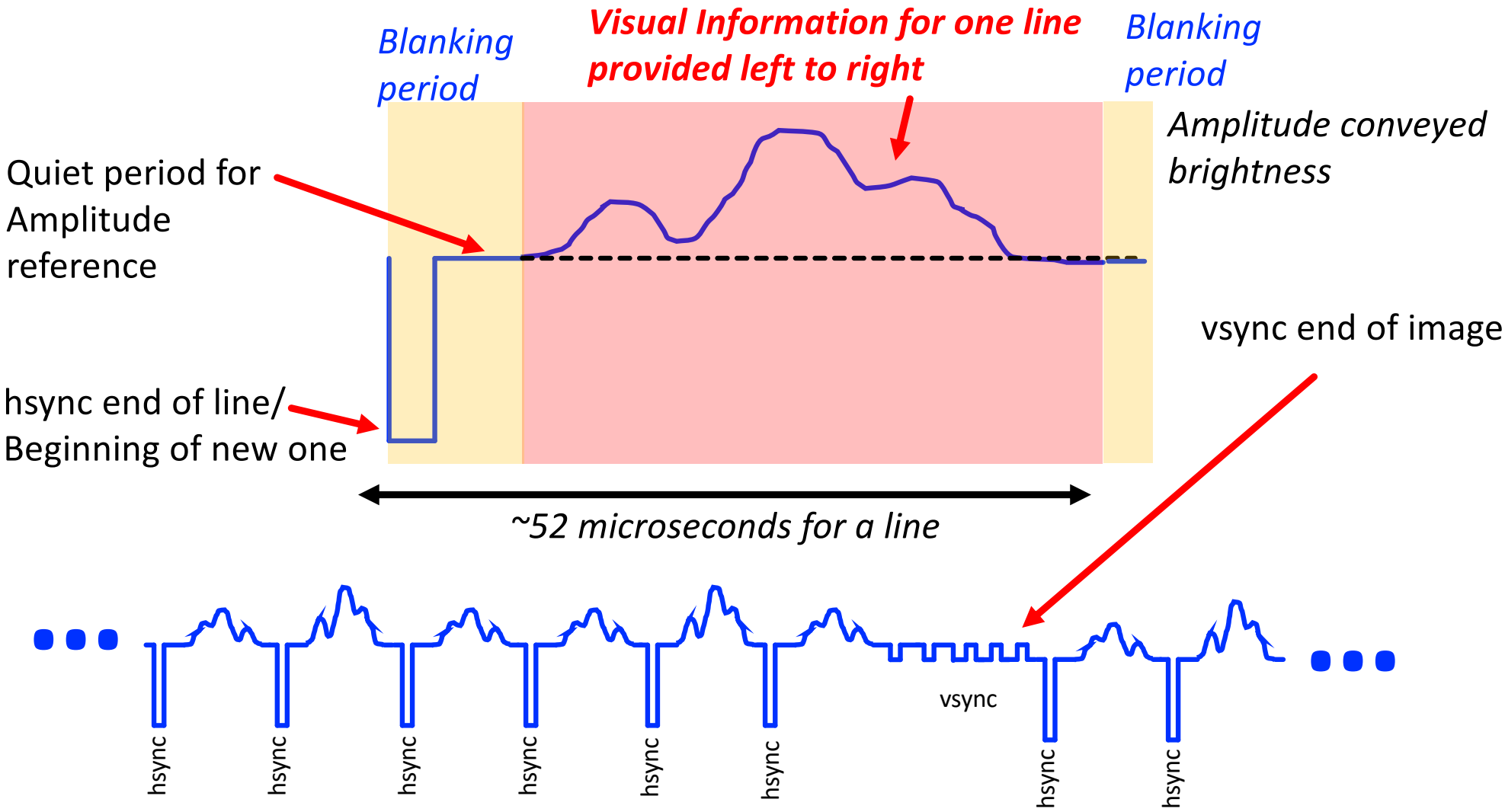


Cathode Ray Tube

http://www.circuitstoday.com/crt-cathode-ray-tube

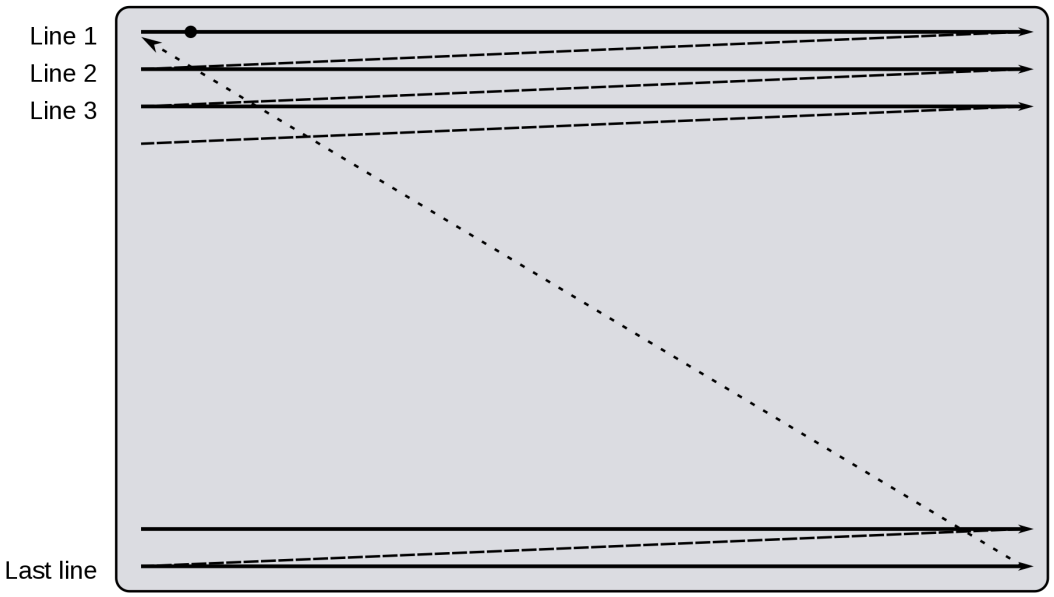
Black and White Video signal

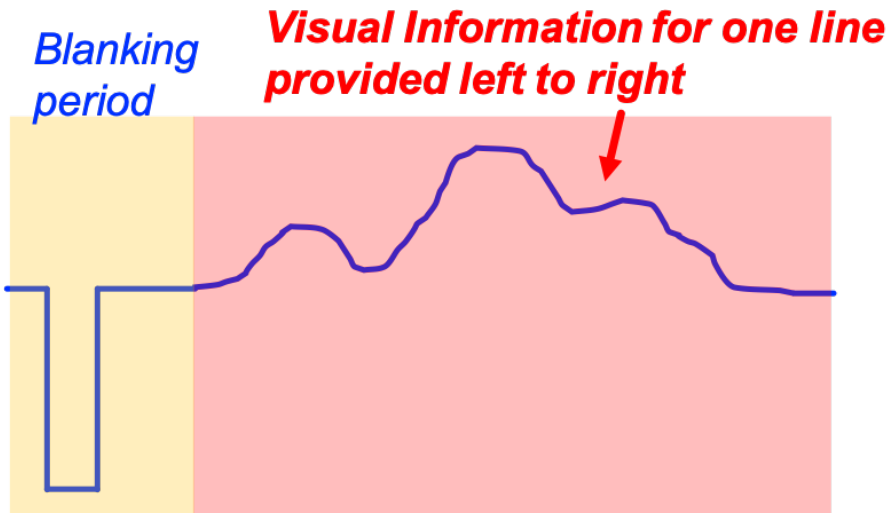
- An **analog** signal conveying luminance (brightness) and synchronization controls (end of line, end of frame)



Controls in Action

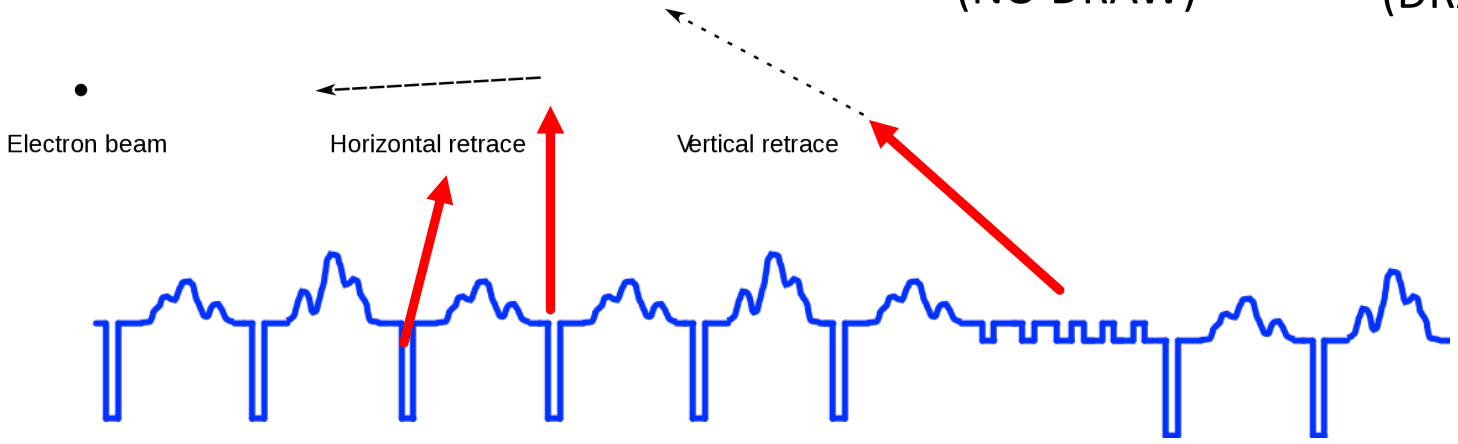
- Signal completely controls beam location and intensity!





BEAM OFF
(NO DRAW)

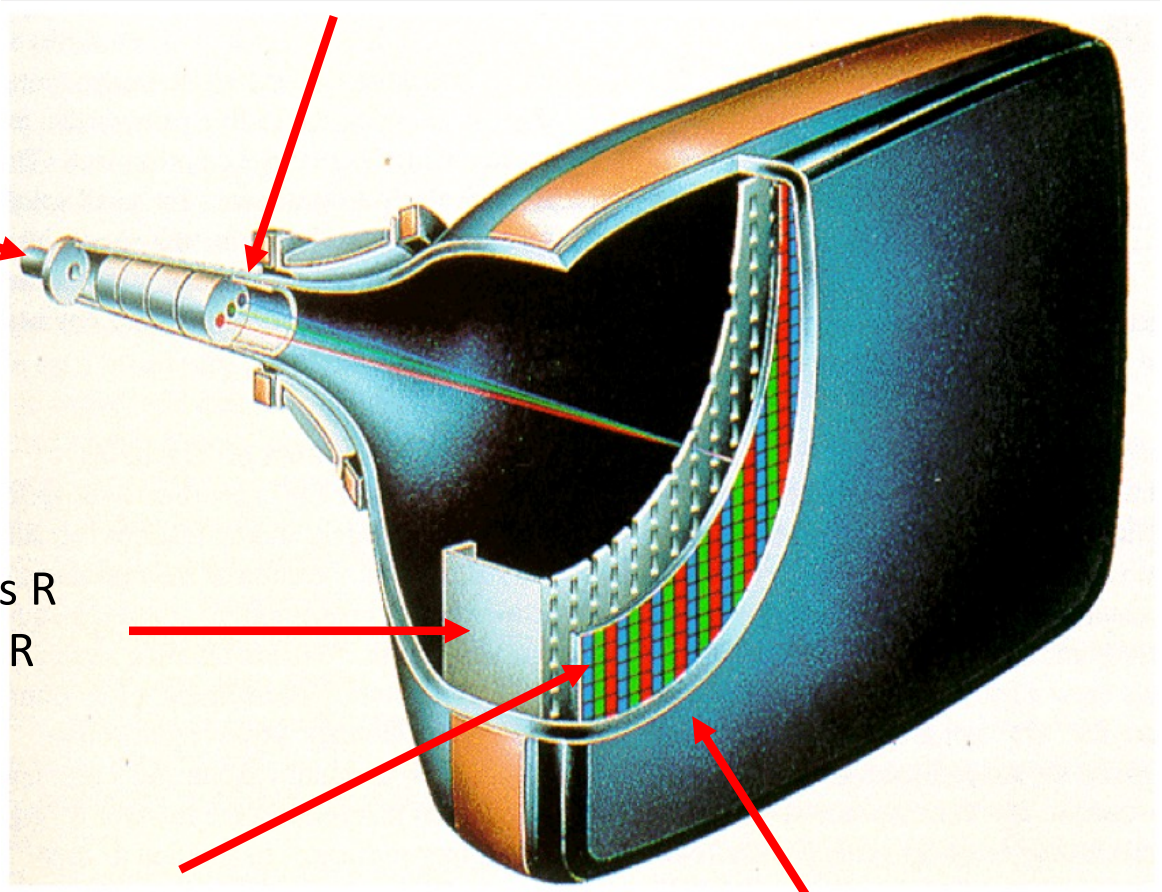
BEAM ON
(DRAW TIME)



Color Cathode Ray Tubes Appear

One shared set of deflection coils to sweep all three beams together

Cathode: separate beams for R, G and B



Shadow mask: ensures R beam only illuminates R pixels, etc.

Three different phosphor screens

Anode

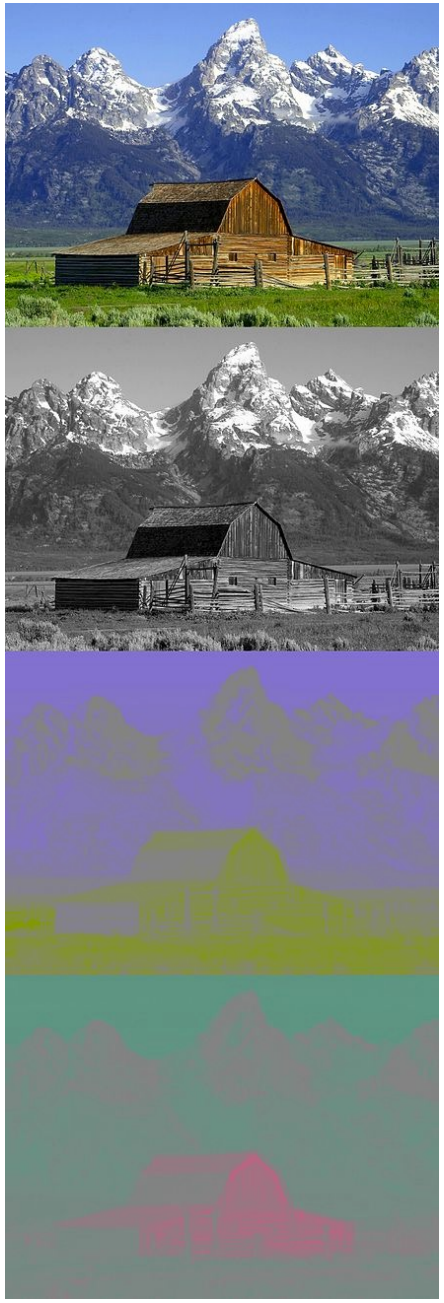
Source: PixTech

How to Upgrade video standards, but let black and white displays still work?

- Color TV invented in 40's but took until 70's for color TV to surpass B&W TV in sales
- How do you do it? Can't send out R, G, and B signals since old TVs won't know what that is
- Still must send out old signal
- Remap our 3D RGB color space into something else!

YCrCb (sometimes YUV)

- Color space composed of three values:
  - Y: Luminance
  - Cr: Red Chrominance
  - Cb: Blue Chrominance
- Together they can represent the full color space



Full color

Y

Cb

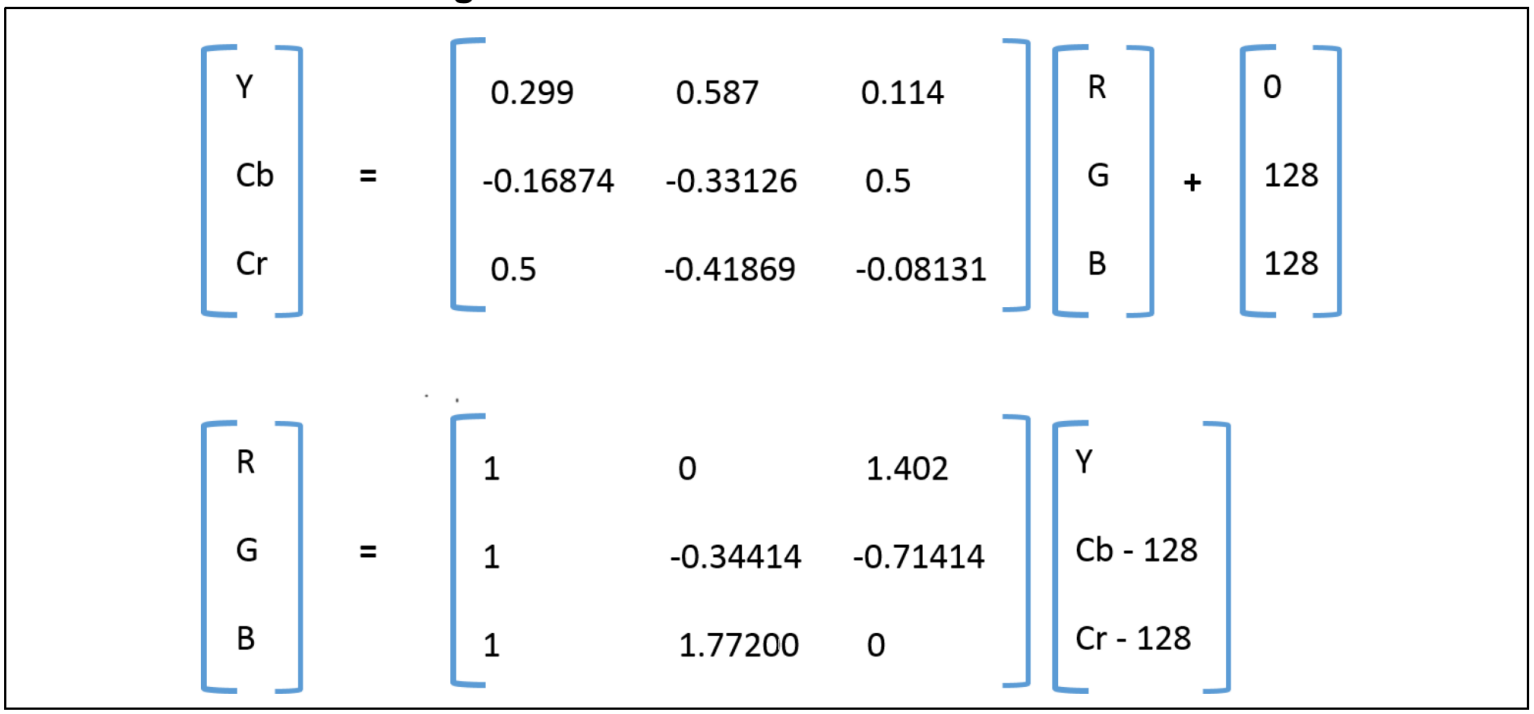
Cr

https://en.wikipedia.org/wiki/YCbCr

YCrCb ←→ RGB

- Just one 3-tuple to another (linear algebra)

Figure 1. YCbCr/RGB color conversion

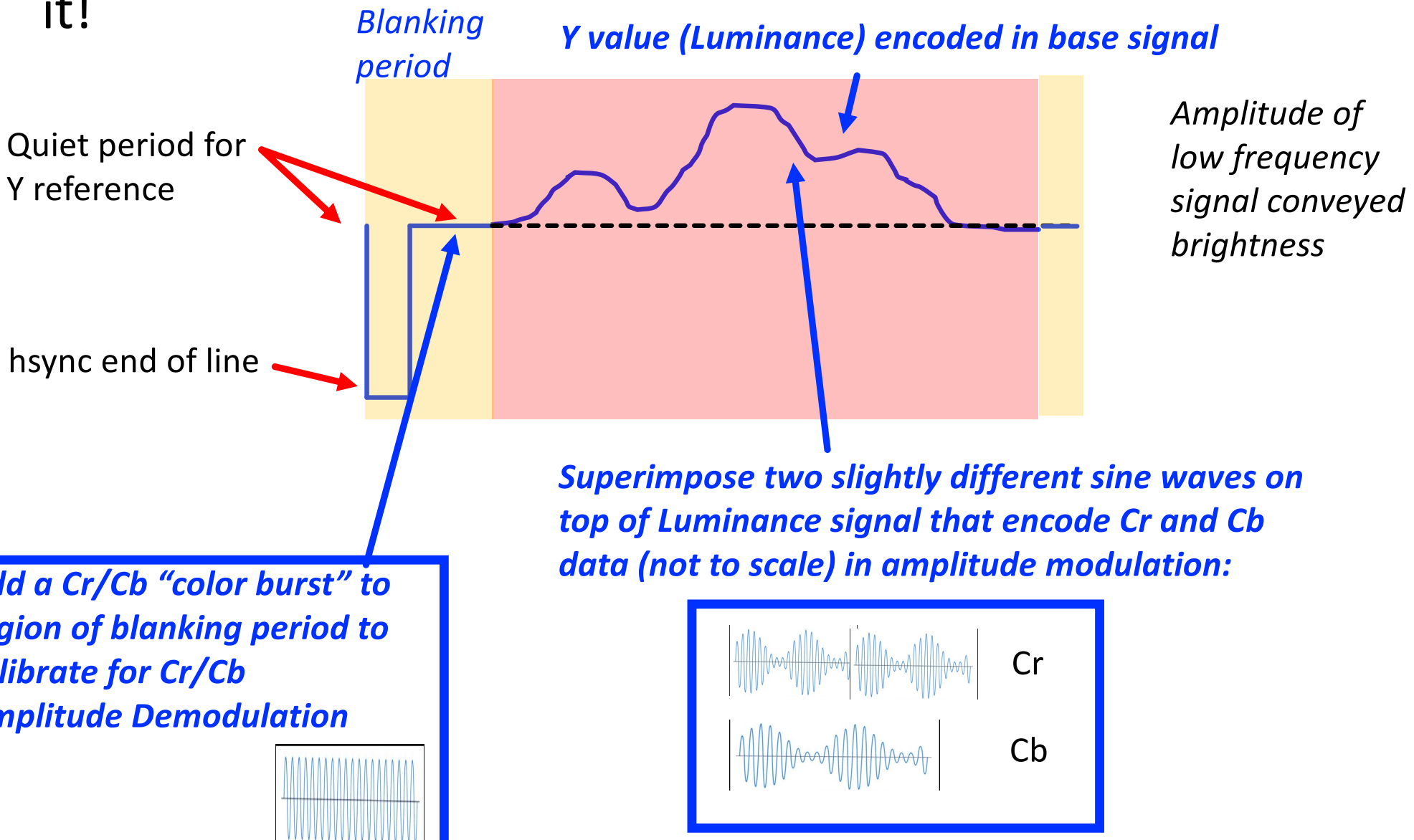


YCrCb ←→ RGB

- 8-bit data
  - R = 1.164(Y – 16) + 1.596(Cr – 128)
  - G = 1.164(Y – 16) – 0.813(Cr – 128) – 0.392(Cb – 128)
  - B = 1.164(Y – 16) + 2.017(Cb – 128)
- 10-bit data
  - R = 1.164(Y – 64) + 1.596(Cr – 512)
  - G = 1.164(Y – 64) – 0.813(Cr – 512) – 0.392(Cb – 512)
  - B = 1.164(Y – 64) + 2.017(Cb – 512)
- Implement using
  - Integer arithmetic operators (scale constants/answer by 2^{11})
  - 5 BRAMs (1024x16) as lookup tables for multiplications

Color Analog Video signal

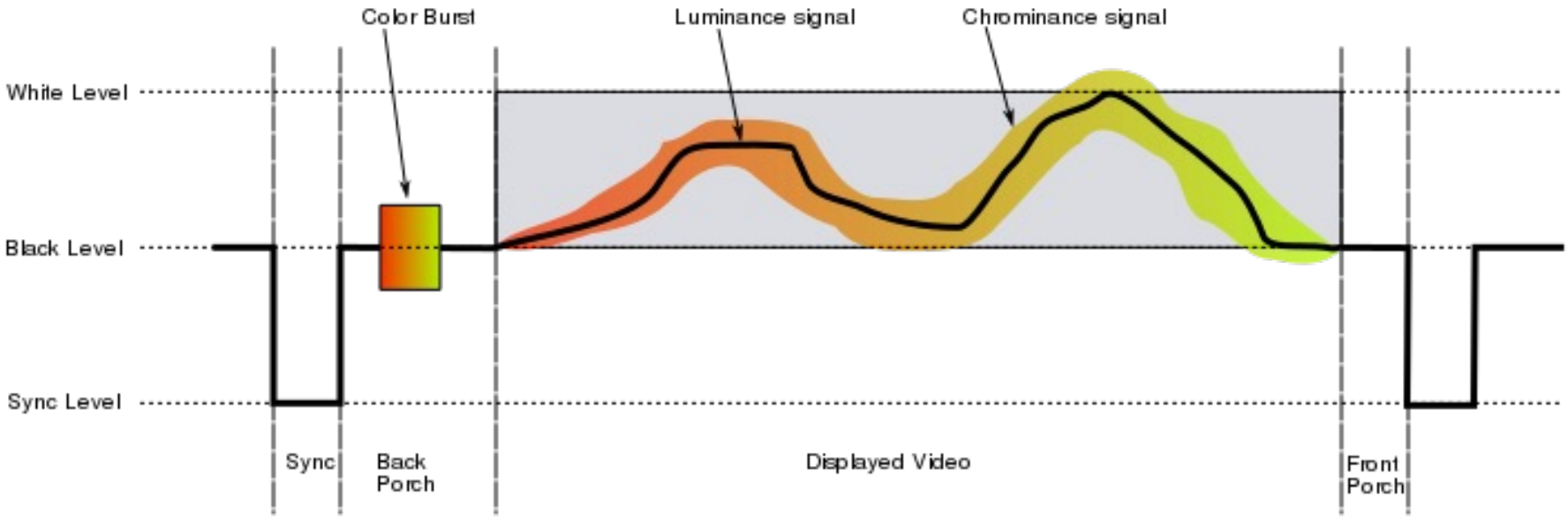
- Keep signal the same as before but add other stuff to it!



Composite Video Encoding:

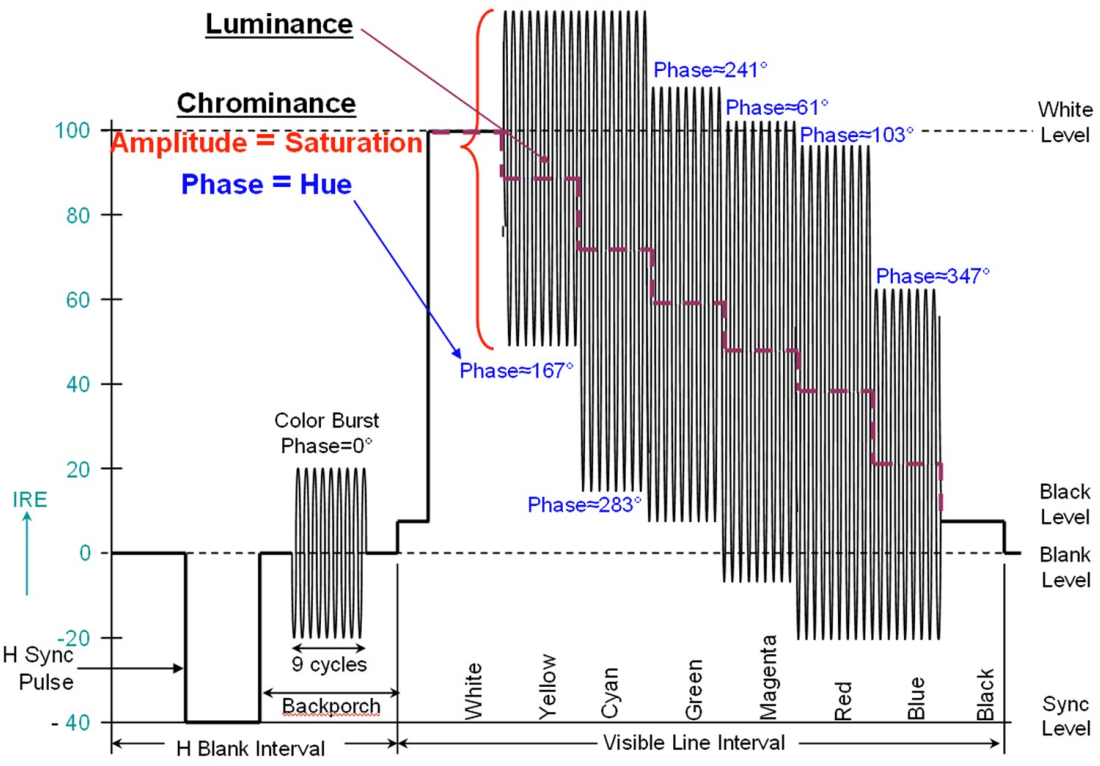
Used for most color TV transmissions and component video up until early 2000's

Use colorburst to remind receiver frequency and amplitudes for interpreting luminance and chrominance signal correctly



Encoding Color

- If you do math out, the two chrominance signals construct/deconstruct to form a signal where:



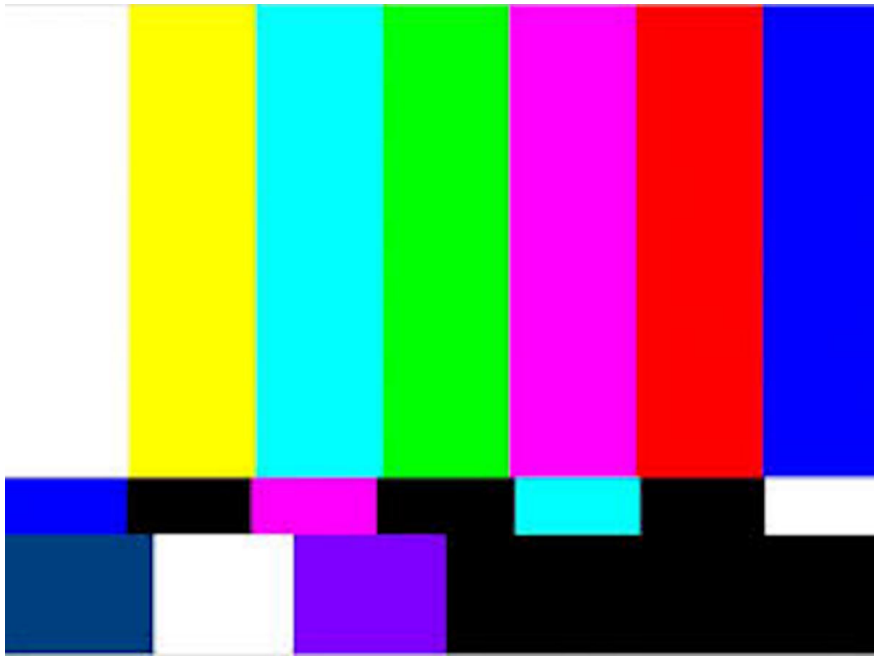
- Amplitude is **Saturation**
- Phase is **Hue**
- **Luminance** is low-freq original value

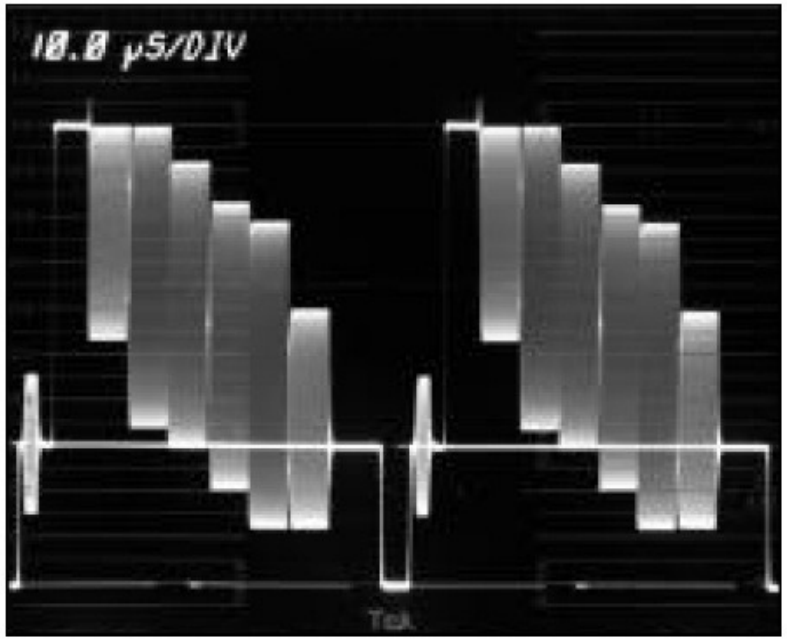
- Hue, Saturation, Luminance (HSL) is a cylindrical color space that is used a lot!

https://www.eetimes.com/document.asp?doc_id=1272387#

NTSC*: Composite Video Encoding

Captures on a Scope





Two horizontal lines shown

Old Labkits work with Cameras that produce composite video out

Two conductors:

- Shield (ground)
- Middle thing (signal)

Component Video Sockets on Virgin Air airplane in 2019

this

Is slightly smaller than

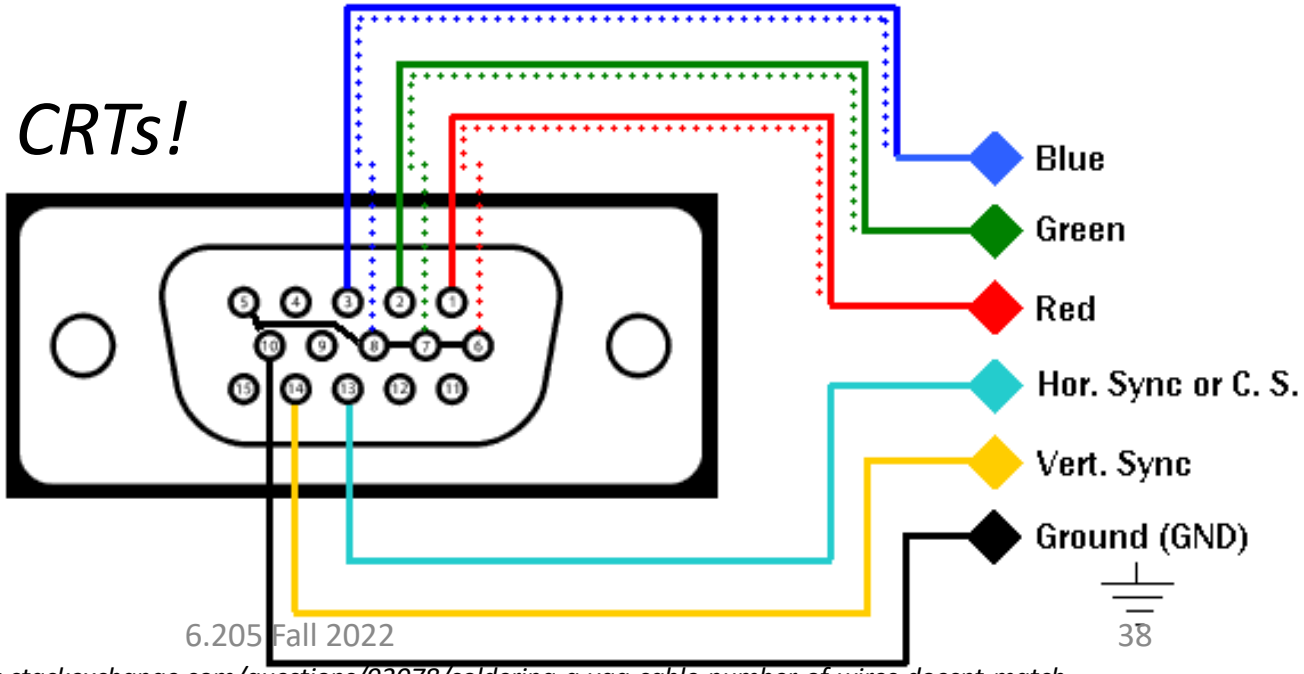
this

Poor engineering.

VGA (Video Graphics Array)

- Development of personal computers motivated a rethink of video display!
- IBM (late 1980s)
- Data conveyed primarily ***analog***
- *Did not have to be reverse compatible with B/W (chose to use RGB as a result)*
- *Used separate wires for different signals (easier)*
- *Still had to deal with CRTs!*
  - *Need blanking!*
  - *Need sync signals!*

DB15 Connector



https://electronics.stackexchange.com/questions/93078/soldering-a-vga-cable-number-of-wires-doesnt-match

VGA Signals

- Similar as Before, but split signals (easier to interpret as human)

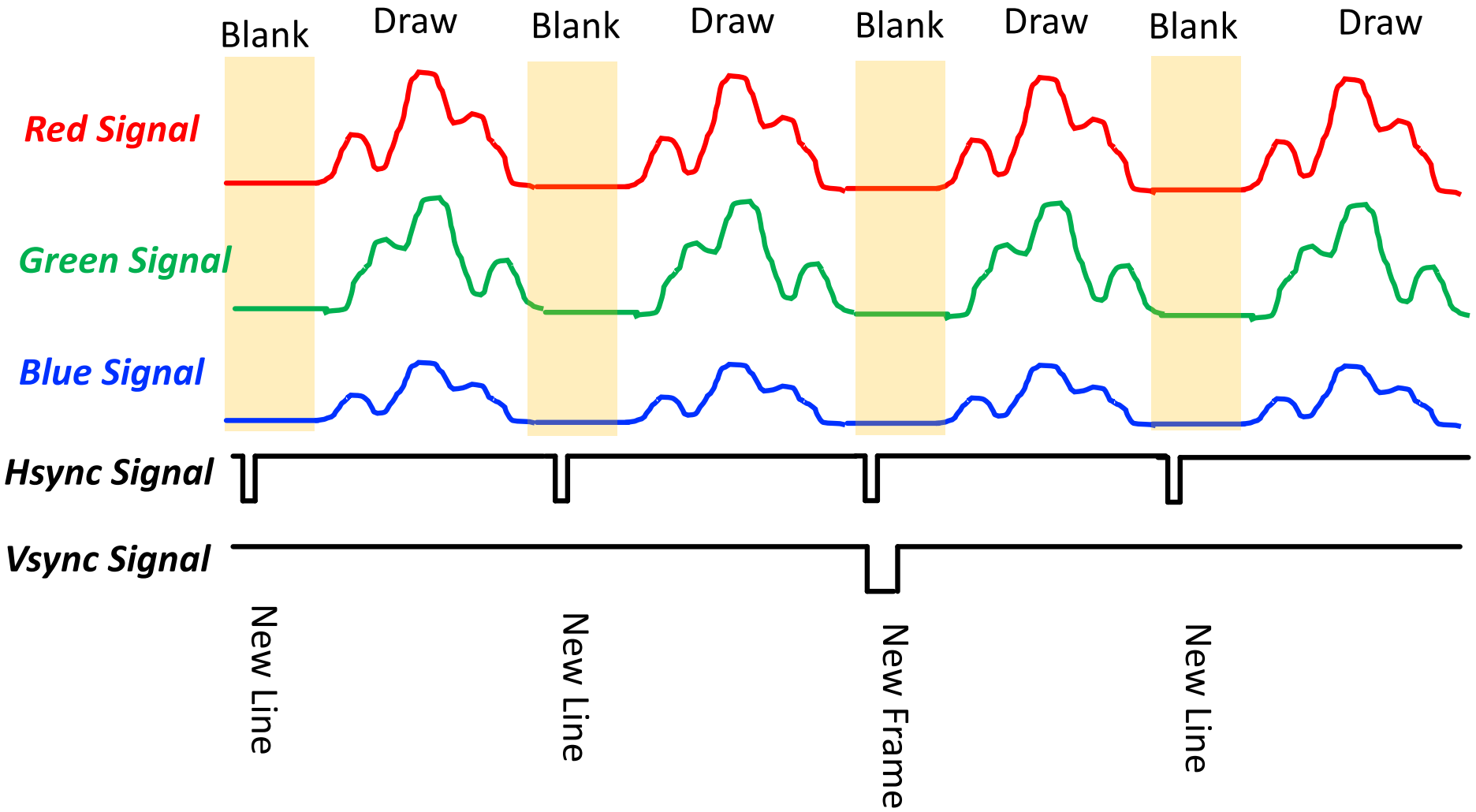


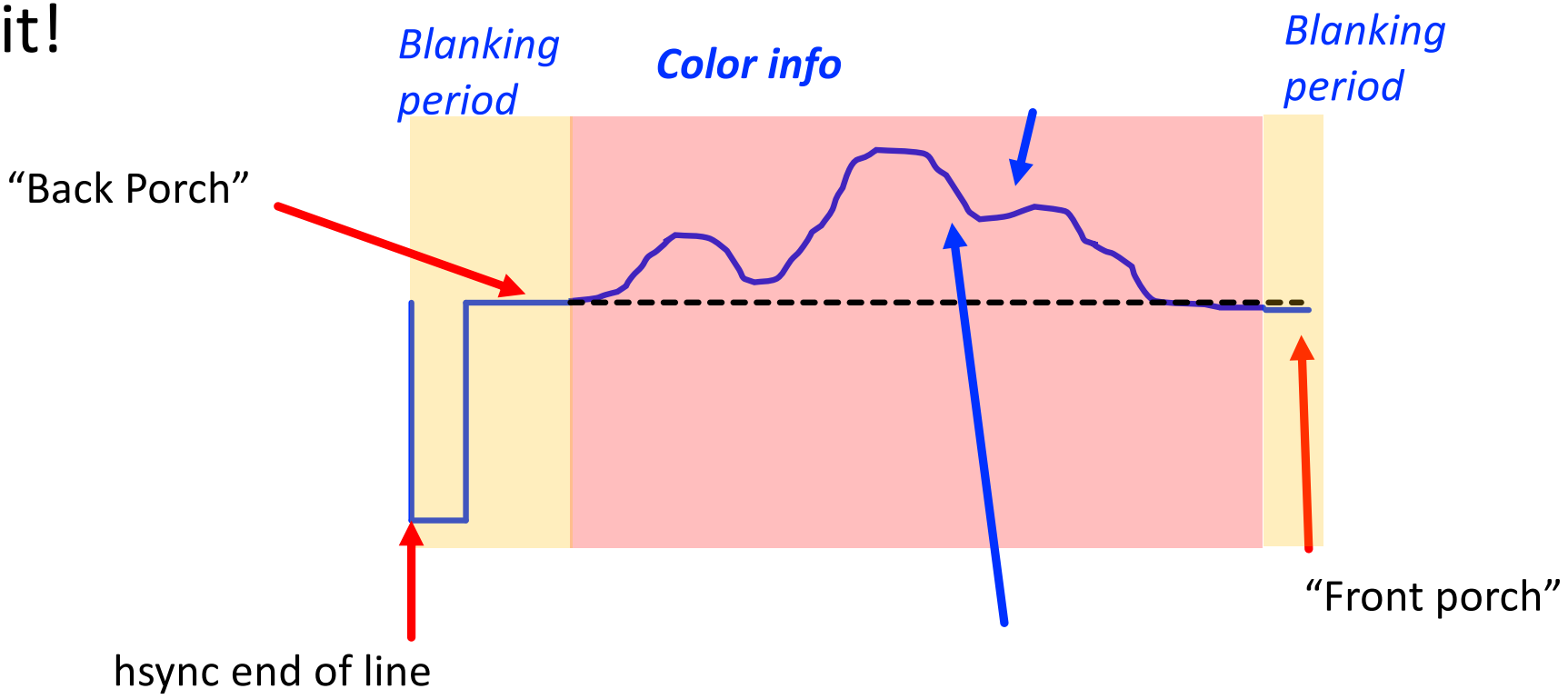
Figure out Display Resolution

Generally need to draw 60 frames per second regardless of resolution(can go faster):

Resolution	Pixels	Aspect Ratio	Products
VGA	640x480	4:3	
SVGA	800x600	4:3	
XGA	1024x768	4:3	iPad, iPad Mini
SXGA	1280x1024	4:3	
HD TV	1920x1080	16:9	
iPhone 6 Plus	1920x1080	16:9	
iPad Retina	2048x1536	4:3	iPad Air, iPad Mini Retina
Macbook Retina	2560x1600	16:10	13” Macbook Pro
Kindle Fire	1920x1200		HDX 7” (3rd Generation)
4K HD TV	3840x2160	16:9	
8K HD TV	7680x4320	16:9	Really expensive TVs

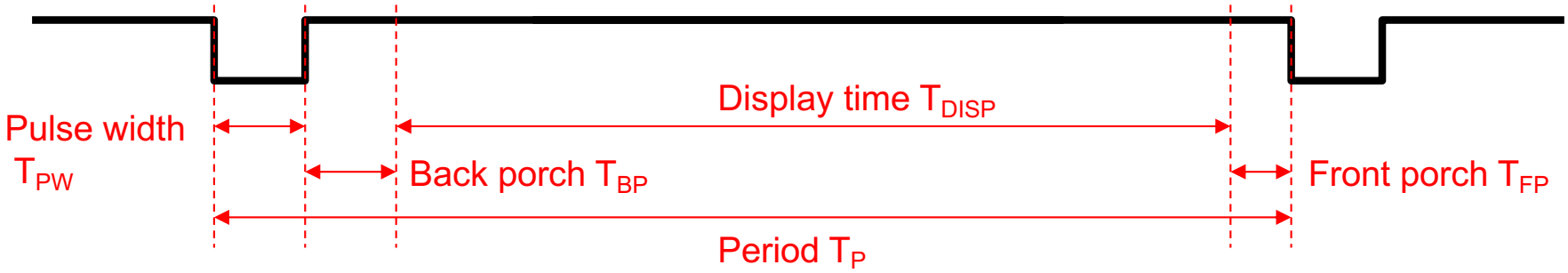
Each line of Video looks like this:

- Keep signal the same as before but add other stuff to it!



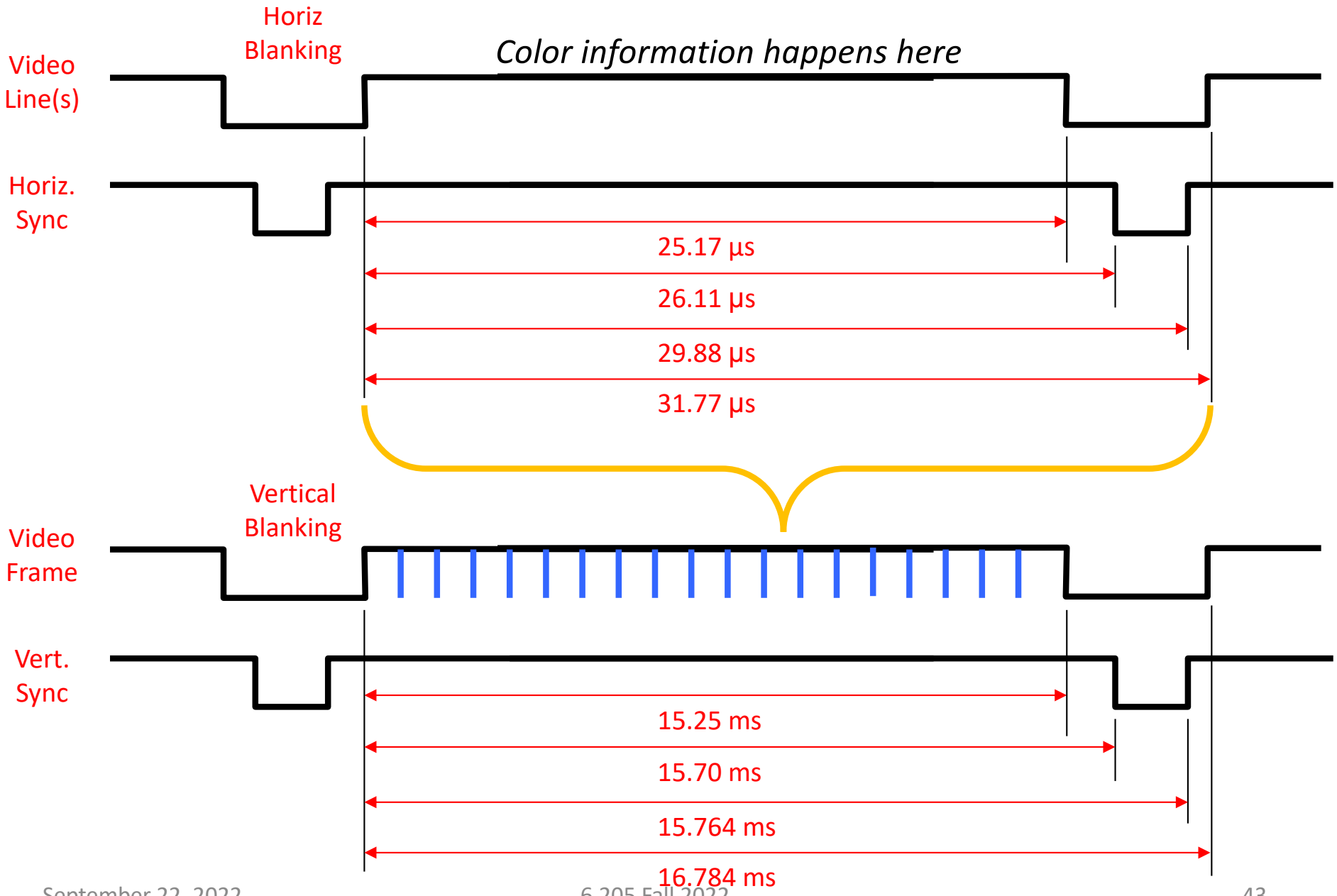
- If we need to draw 60 XGA frames per second, what must the timing of our lines be?
- Timing of our pixels?

Sync Signal Timing



Format		*CLK*	*P*	*PW*	*BP*	*DISP*	*FP*
VGA	HS (pixels)	25Mhz	794	95	47	640	13
	VS (lines)	--	528	2	33	480	13
XGA	HS (pixels)	65Mhz	1344	136	160	1024	24
	VS (lines)	--	806	6	23	768	9

VGA (640x480) Video (for example)



vga module

- Given in Lab03: generates video-compatible signals
- Starting with 65 MHz video clock, generate:
- ***hsync, vsync*** *signals*
- ***blanking*** *signal!*
- *Horizontal and vertical counts (**hcount, vcount**)*

Given in Lab 03

```
module xvga(input vclock_in,
            output reg [10:0] hcount_out,    // pixel number on current line
            output reg [9:0] vcount_out,     // line number
            output reg vsync_out, hsync_out,
            output reg blank_out);

   parameter DISPLAY_WIDTH  = 1024;      // display width
   parameter DISPLAY_HEIGHT = 768;       // number of lines

   parameter  H_FP = 24;                 // horizontal front porch
   parameter  H_SYNC_PULSE = 136;        // horizontal sync
   parameter  H_BP = 160;                // horizontal back porch

   parameter  V_FP = 3;                  // vertical front porch
   parameter  V_SYNC_PULSE = 6;          // vertical sync
   parameter  V_BP = 29;                 // vertical back porch

   // horizontal: 1344 pixels total
   // display 1024 pixels per line
   reg hblank,vblank;
   wire hsyncon,hsyncoff,hreset,hblankon;
   assign hblankon = (hcount_out == (DISPLAY_WIDTH -1));
   assign hsyncon = (hcount_out == (DISPLAY_WIDTH + H_FP - 1));  //1047
   assign hsyncoff = (hcount_out == (DISPLAY_WIDTH + H_FP + H_SYNC_PULSE - 1));  //
   assign hreset = (hcount_out == (DISPLAY_WIDTH + H_FP + H_SYNC_PULSE + H_BP - 1));

   // vertical: 806 lines total
   // display 768 lines
   wire vsyncon,vsyncoff,vreset,vblankon;
   assign vblankon = hreset & (vcount_out == (DISPLAY_HEIGHT - 1));   // 767
   assign vsyncon = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP - 1));  // 771
   assign vsyncoff = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP + V_SYNC_PULSE -
   assign vreset = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP + V_SYNC_PULSE + \

   // sync and blanking
   wire next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
   always_ff @(posedge vclock_in) begin
      hcount_out <= hreset ? 0 : hcount_out + 1;
      hblank <= next_hblank;
      hsync_out <= hsyncon ? 0 : hsyncoff ? 1 : hsync_out;  // active low

      vcount_out <= hreset ? (vreset ? 0 : vcount_out + 1) : vcount_out;
      vblank <= next_vblank;
      vsync_out <= vsyncon ? 0 : vsyncoff ? 1 : vsync_out;  // active low

      blank_out <= next_vblank | (next_hblank & ~hreset);
   end

endmodule
```

VGA (1024x768) Sync Timing

```
// assume 65 Mhz pixel clock

// horizontal: 1344 pixels total
// display 1024 pixels per line
assign hblankon = (hcount == 1023);  // turn on blanking
assign hsyncon = (hcount == 1047);   // turn on sync pulse
assign hsyncoff = (hcount == 1183);  // turn off sync pulse
assign hreset = (hcount == 1343);    // end of line (reset counter)

// vertical: 806 lines total
// display 768 lines
assign vblankon = hreset & (vcount == 767);   // turn on blanking
assign vsyncon = hreset & (vcount == 776);    // turn on sync pulse
assign vsyncoff = hreset & (vcount == 782);   // turn off sync pulse
assign vreset = hreset & (vcount == 805);     // end of frame
```

Has other timing specs specified for different types of displays!

Modern Displays and Technologies

VGA is dead, Joe. My computer only has HDMI and a Display Port. This is irrelevant

Display Types

- Emissive Display
  - Organic Light Emitting Diode (OLED) Displays
  - Liquid Crystal Display (LCD)
    - requires backlight source,
    - constant power
  - Cathode Ray Tube (CRT)

Back in Time

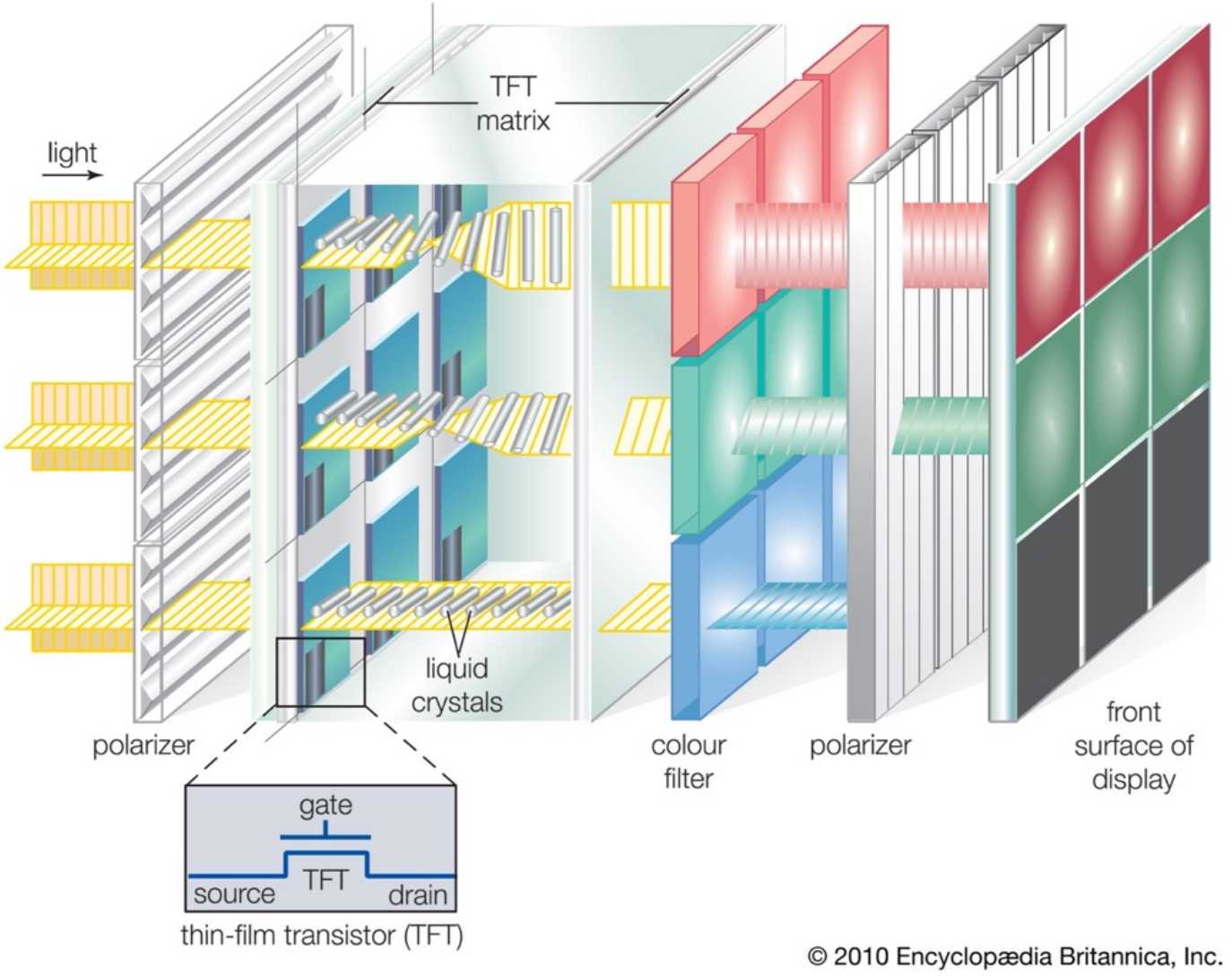
- Reflective Display
  - Electrophoretic Display (E-Ink)*
    - Ultra Low Power – displays are bi-stable, drawing power only when updating the display.
    - Viewable in sunlight – ambient light reflected from display
  - Liquid Crystal Display (LCD)
    - I'm talking old-school calculator style here

Back in Time

*Prof Joseph Jacbson, MIT

TFT LCD

Used to be Cold Cathode
Now almost always white LEDs



liquid crystal display: active-matrix TFT liquid crystal display. Art. *Encyclopædia Britannica Online*. Web.

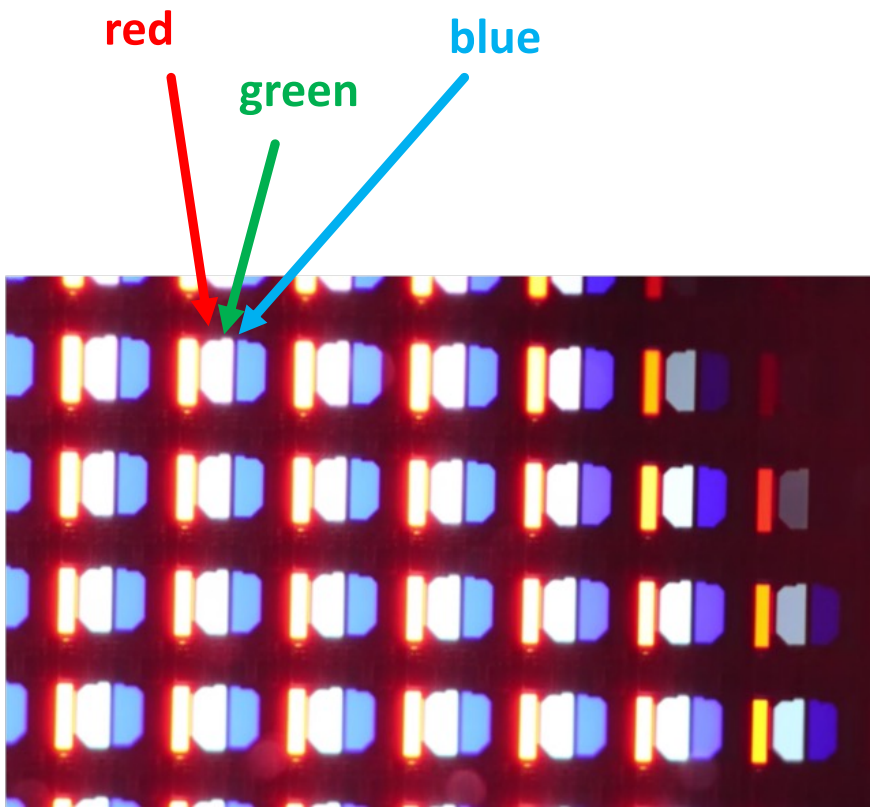
TFT (Thin-Film Transistors)

- Older Technology:
- Make a display:
  1. Gigantic white backlight (polarized)
  2. Gigundous array of voltage-variable polarizers (TFTs with Liquid Crystals) (let light through at rest)
  3. One TFT for each color (RGB), three per pixel
- Want black pixel? Turn TFT fully on to block light getting through

Organic Light Emitting Diodes

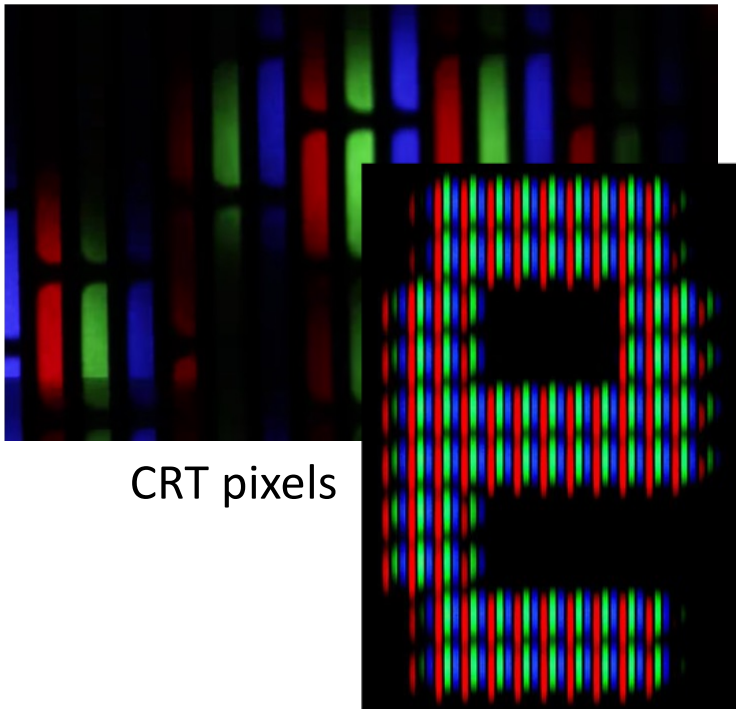
- Newest Technology
- Conceptually maybe the simplest/ideal way to do a display
  1. Gigundous array of RGB LEDs
  2. Control RGB amt. at each point
  3. Profit

1. Want black pixel? Just don't turn on LED



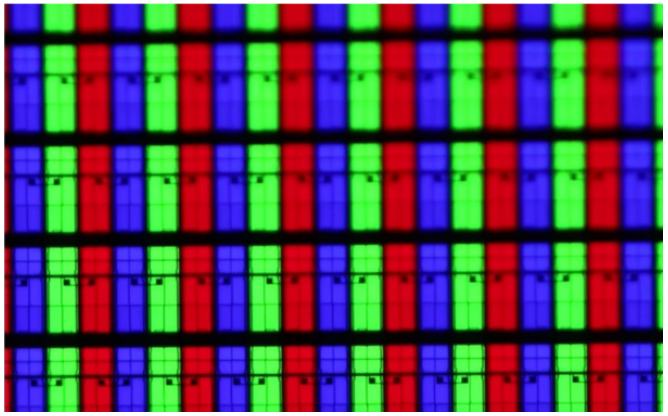
***Green saturated in this image**

All Color Displays use RGB Pixels

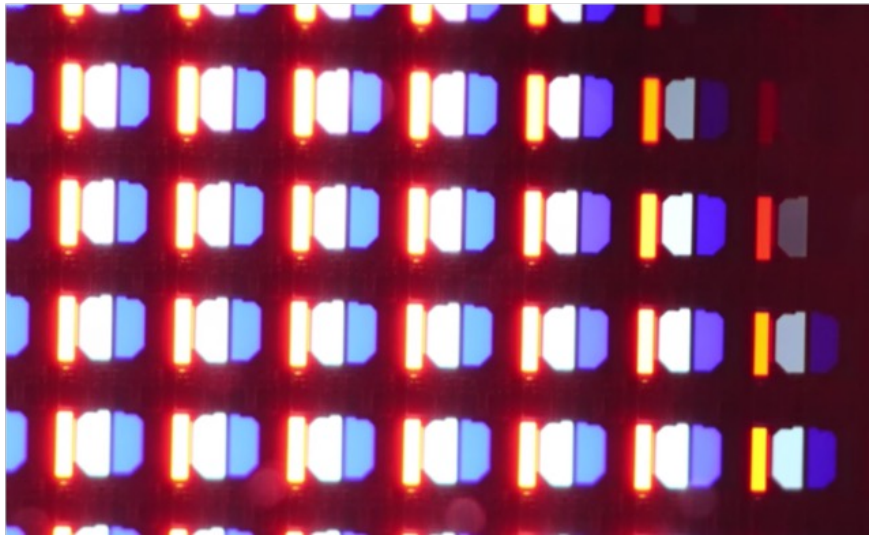


CRT pixels

CRT pixels



TFT LCD pixels



OLED pixels

Slo-Mo Guys

https://www.youtube.com/watch?v=3BJU2drrtCM

- Video Locations:
  - CRT @2:13
  - TFT LCD @ 7:58
  - OLED @ 10:50

- Whole video is a good watch

DVI (Digital Video Interface)

- 1998ish
- Backwards compatible with VGA to an extent (supposed to support analog)
- Sends data digitally over twisted pairs in high-level structure similar to VGA

DB15 Connector

Twisted pairs for data, power, sensing

Analog channels

HDMI

- It all starts with the cable and connector

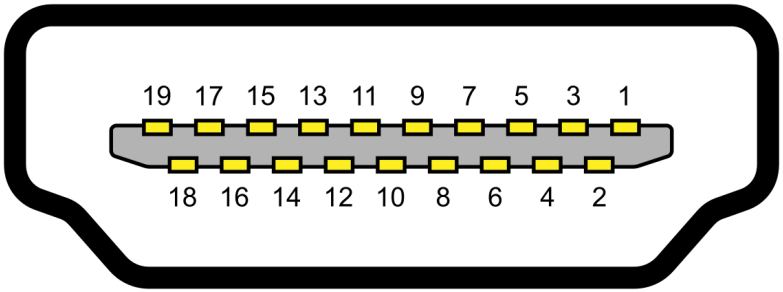


Table 1. HDMI

Pin Number	Assignment
1	Data2+
2	Data2 shield
3	Data2-
4	Data1+
5	Data1 shield
6	Data1-
7	Data0+
8	Data0 shield
9	Data0-
10	Clock+
11	Clock shield
12	Clock-
13	CEC
14	Not connected
15	SCL
16	SDA
17	Ground
18	+5V
19	Hot-plug detect

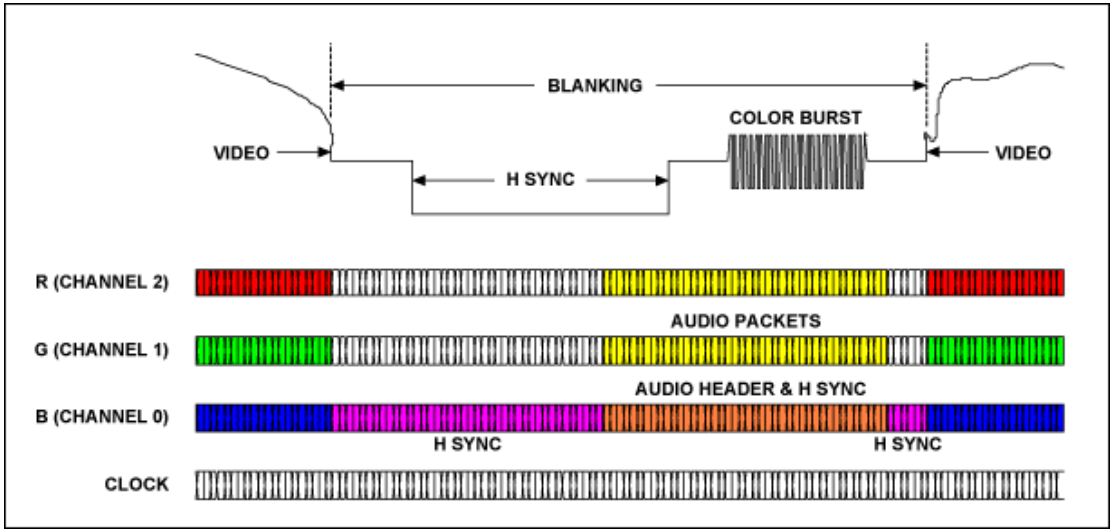
- You've got three pairs* of wires that carry color
  - Channel 0: Blue
  - Channel 1: Green
  - Channel 2: Red
- Clock Channel
- Few other wires:
  - Resolution info
  - CEC (control things)
  - Power

*each group is a differential signal pair and shield

https://www.maximintegrated.com/en/app-notes/index.mvp/id/4306

Color Information

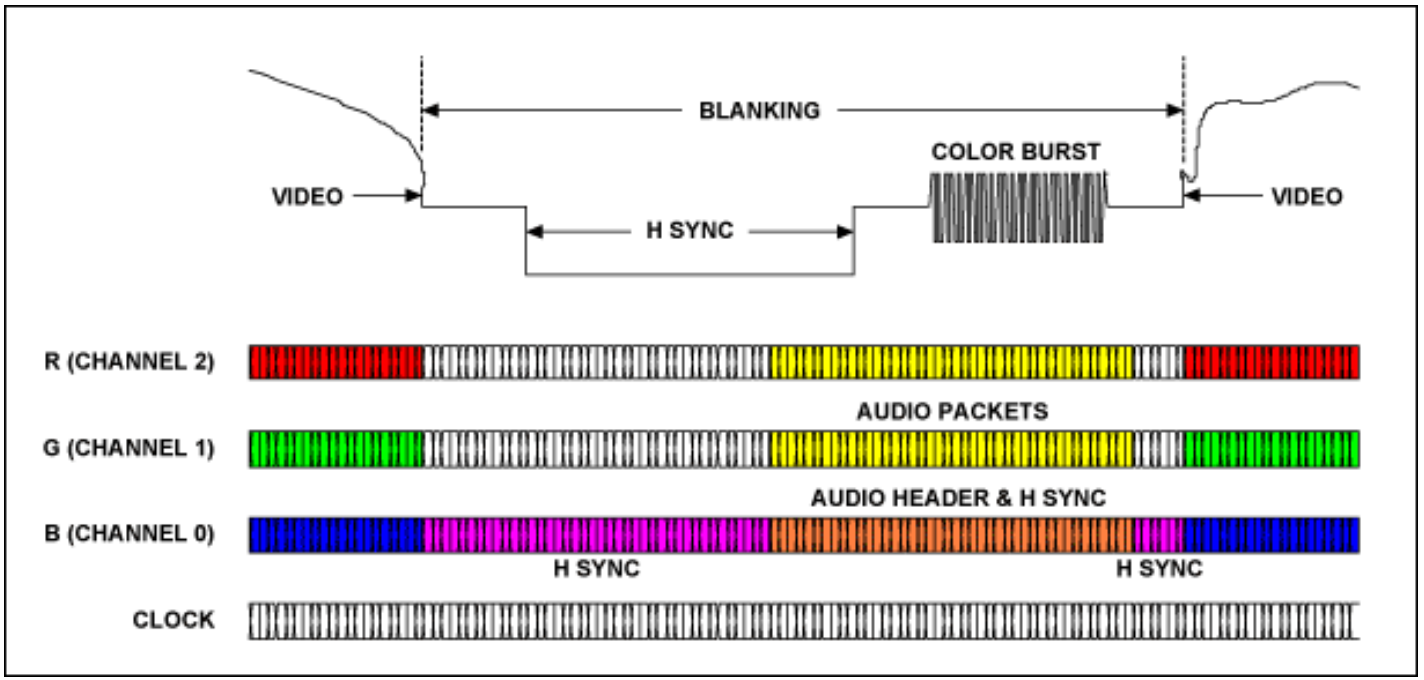
- Sent as serialized data in 10-bit frames using TMDS (Pset 3 and on…)
- One color per pair of wires (red, green, blue wires)
- The blue channel also carries blanking/hsync/vsync info:
  - Encodes those using four 10 bit reserved values:
  - (H = 0, V = 0): 1101010100
  - (H = 1, V = 0): 0010101011
  - (H = 1, V = 0): 0101010100
  - (H = 1, V = 1): 1010101011



One pixel of information per clock cycle (clock is 1/10 bit rate)

Audio Information

- During blanking period (when no color needs to be conveyed), there's unused clock cycles on the color lines.
- Shove audio into that region
- Blanking region works out to be about 64 pixels worth of time (64 clock cycles) per line



Traditional Video

HDMI

Audio

- With a screen refresh rate of 60Hz…
- 1080 lines per screen…
- 64 pixels per line (blanking time we have to play with)…
- and 8 bits (of info) per pixel for an HDTV signal…
- The maximum audio information bit rate we could send is:

  $= 60 \times 1080 \times 64 \times 8 = 33.1776\text{Mbps}$

  This data rate is more than sufficient to carry any multichannel high-quality audio signals
- (Stereo CD-quality Audio needs 1.411Mbps as a reference)
- Plenty of leftover bandwidth for spyware, malware, etc

https://www.maximintegrated.com/en/app-notes/index.mvp/id/4306

Conclusion

- HDMI is heavily based off of VGA and is therefore easy to convert.
- Designing for VGA is directly portable (and oftentimes will work without change) for modern video processing (lab kit)
  - Left→Right, Top→Bottom Raster pattern
  - RGB specification of each pixel
  - Blanking (pause periods) where you don't draw and can potentially do heavier calculations if needed!
- Just use different interface circuits and watch your timing!

FPGA board generates VGA

- Can give you adapters to convert to HDMI if you'd like so you can use HDMI monitors where you live

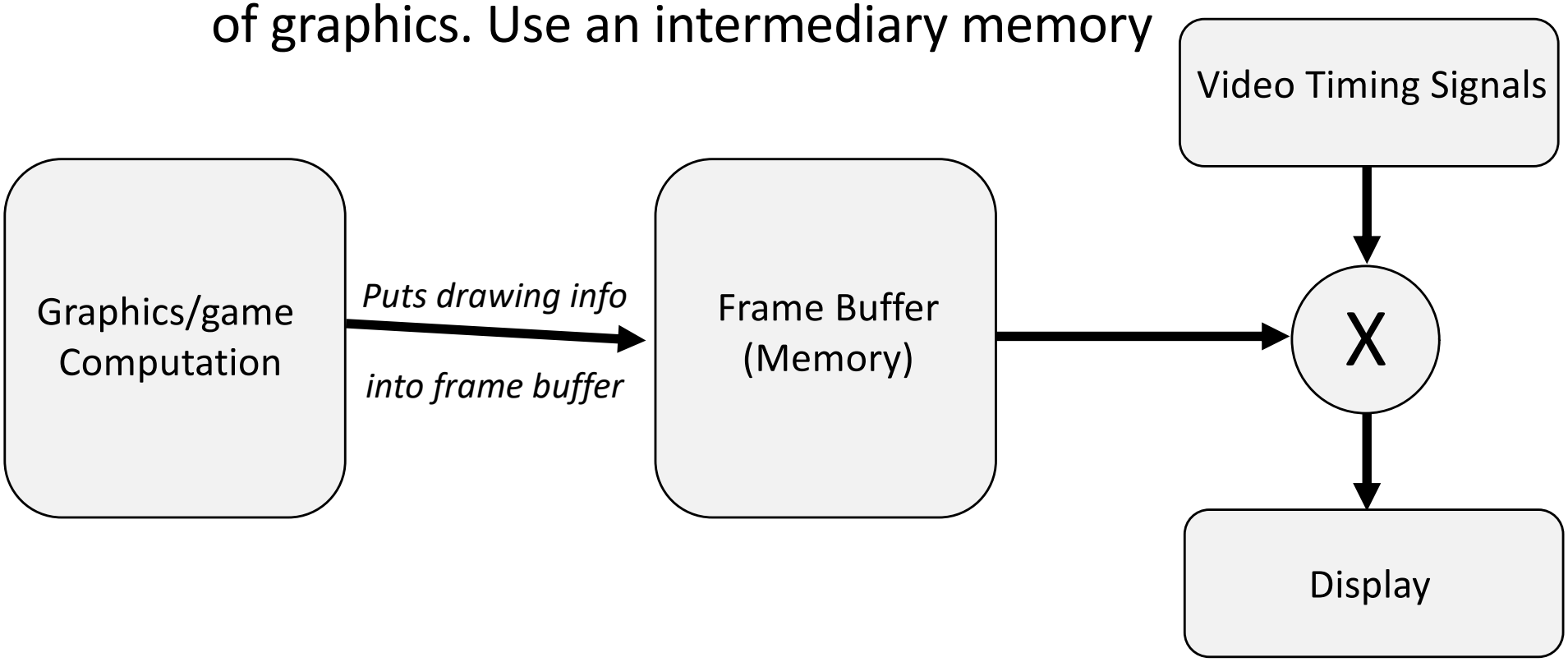
- FPGAs are also fast enough where you can actually generate your own HDMI signal as well (though not in Lab 03 anyways)

Generating Video on the FPGA

Lab 03 Pong:

Two General Ways to Produce Video:

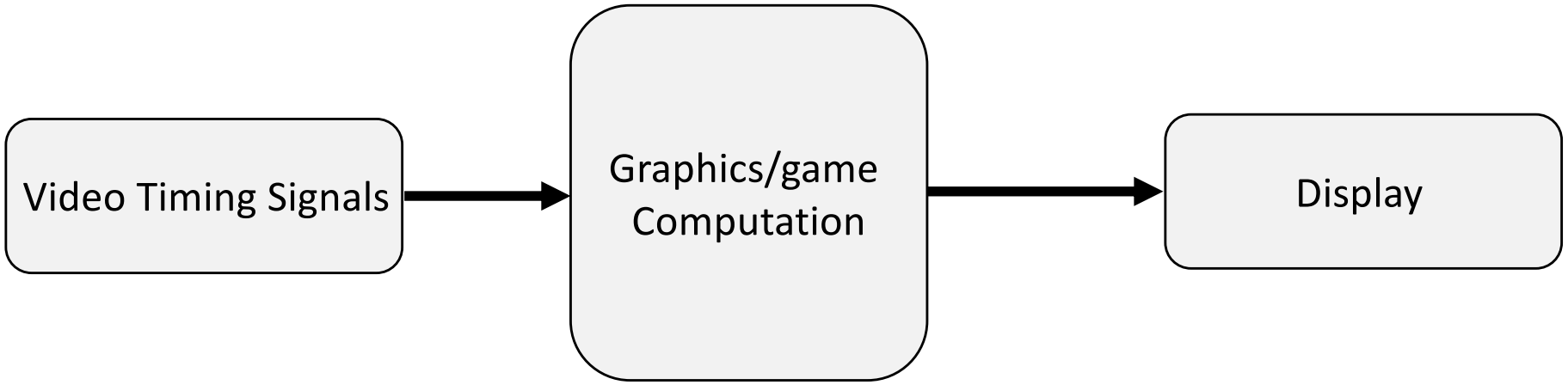
- Way One: Frame Buffer:
  - Separate computation of drawing from actual rendering of graphics. Use an intermediary memory



More modern way of doing it (need lots of memory though!)

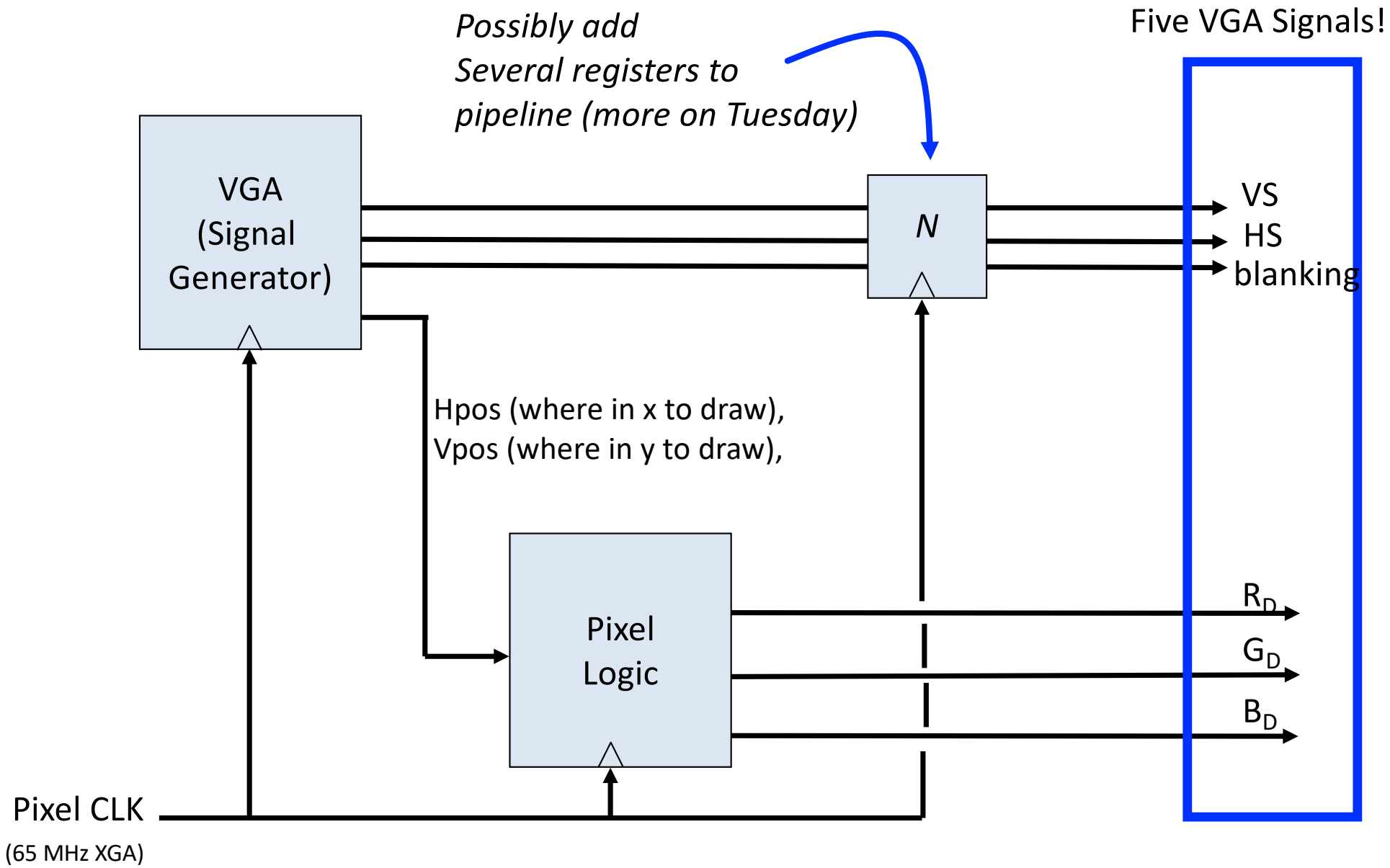
Two General Ways to Produce Video:

- Way Two: “Racing the Beam”:
  - Have everything run off of the video timing signals and compute the value of the pixel just in time when needed!



- *How a lot of early video games and other things were done (and other niche applications today).*
- *All computation (game logic and render logic) must be done on the clock cycle that it is needed*

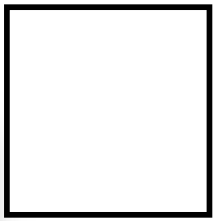
Lab 03 Setup (Racing the Beam)



Examples of pixel logic:

Whole Screen is White:

```
always_ff @(posedge pixel_clk) begin
   if (vblank | (hblank & ~hreset))
       rgb <= 12'h0;
   else
       rgb <= 12'hF_F_F;  //Nexys has 12 bit color
end
```

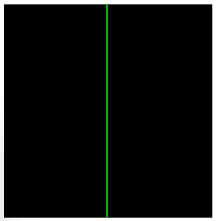


RGB	*Color*
000	black
001	blue
010	green
011	cyan
100	red
101	magenta
110	yellow
111	white

Normalized color scale

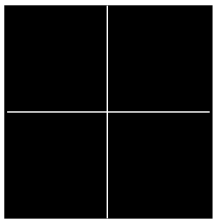
Draw green vertical line at horizontal spot 500:

```
always_ff @(posedge pixel_clk) begin
   if (h_count==500)
       rgb <= 12'h0_F_0; //draw green line
   else
       rgb <= 12'h000;
end
```



Draw white crosshair at (500,500)

```
always_ff @(posedge pixel_clk) begin
   if (h_count==500 || v_count==500)
       rgb <= 12'hF_F_F; //draw white cross-hair
   else
       rgb <= 12'h000;
end
```

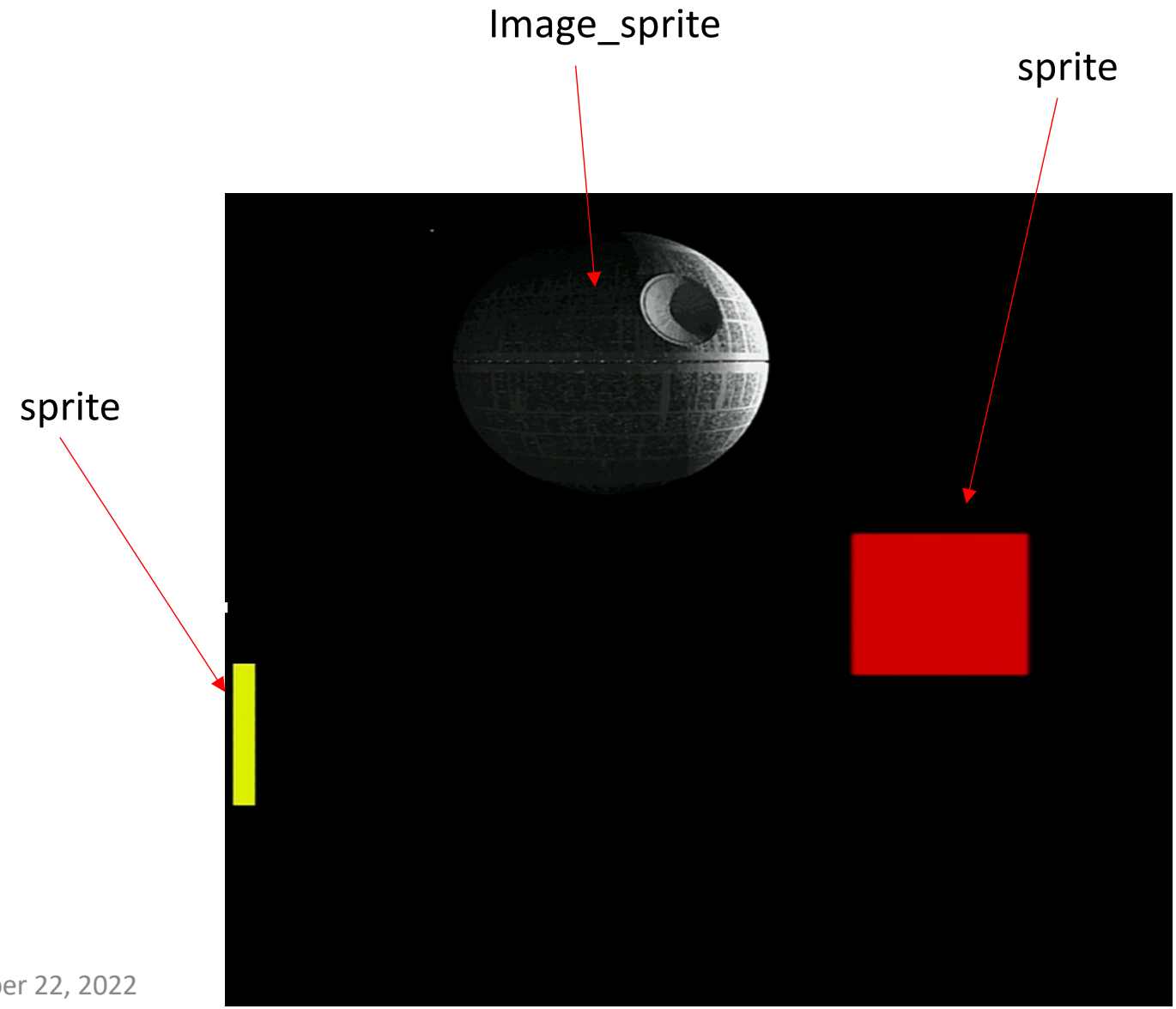


Examples of Pixel Logic

- From Lab 03:
- Draw a white pixel if within a set of rectangular bounds!

```
//////////////////////////////////////////////////////////////////////
//
// blob: generate rectangle on screen
//
//////////////////////////////////////////////////////////////////////
module blob
   #(parameter WIDTH = 64,            // default width: 64 pixels
               HEIGHT = 64,           // default height: 64 pixels
               COLOR = 12'hFFF)  // default color: white
   (input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);

   always_comb begin
      if  ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) &&
           (vcount_in >= y_in && vcount_in < (y_in+HEIGHT)))
        pixel_out = COLOR;
      else
        pixel_out = 0;
   end
endmodule
```

Sprites and Image Sprites:



In Lab03…An image

- We'll build up a memory infrastructure to lookup and render an image of the PopCat or whatever you want

Color Maps

- Images are large and take up lots of memory
- Want to save space, and store image using less memory
- Many images don't express every one of the 2^{24} "true" colors.
- Why waste the space storing an unused possibility?
- So pick the *N* most popular and only display them:
  - You can store them using $\text{ceil}(\log_2(N))$ bits (save space)
  - Then use a color table to look up what full color (24 bit value) that corresponds to!

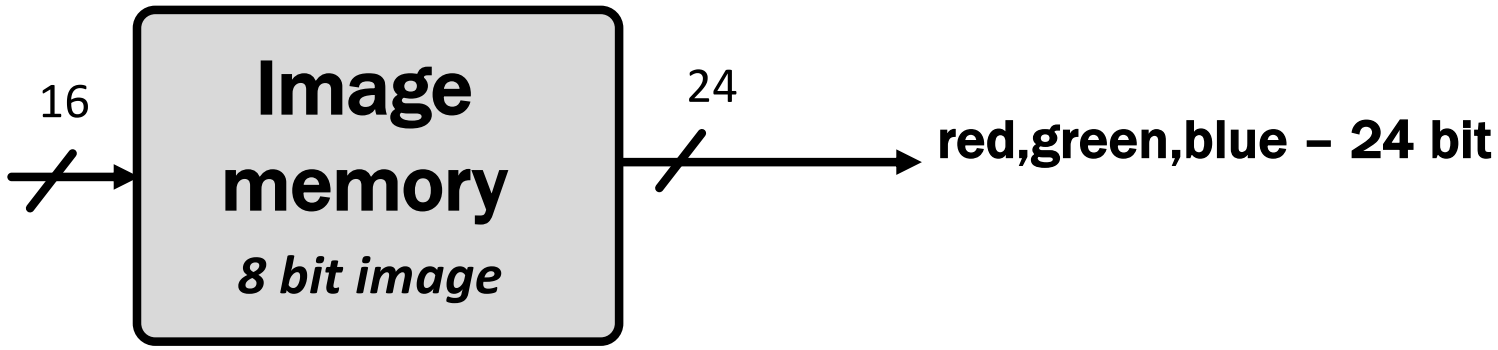
4 bit – 16 colors

8 bit – 256 colors

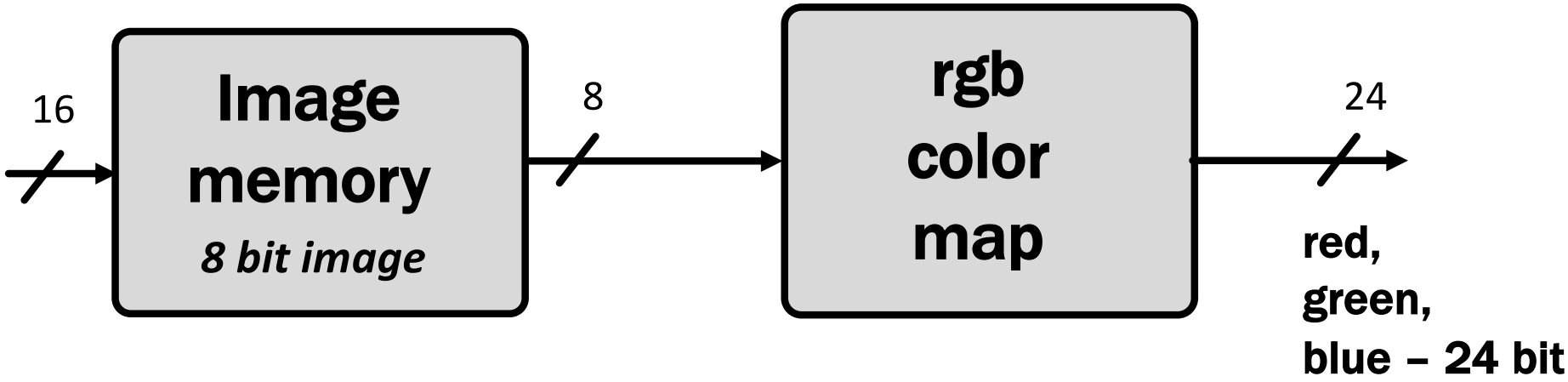
24 bit – 16M colors

Raw Image

24 bit – 16M colors



Color Lookup Table



8 bit – 256 colors

Storage Savings

24 bit – 16M colors

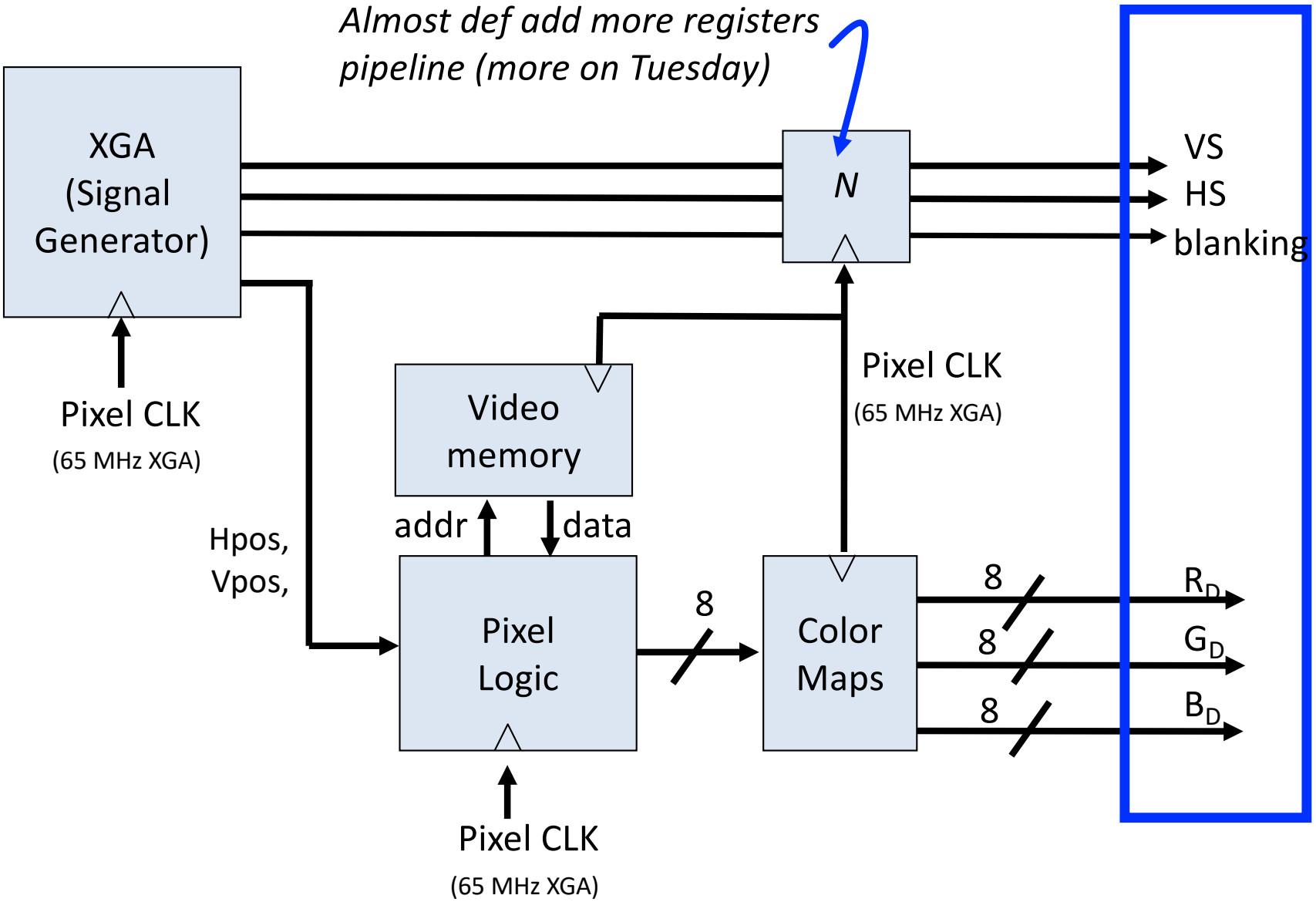
256 X 256 image @ 24 bits per pixel is:
256 X 256 X 24 bits = 1.572 Mbits
(196.6 kBytes)

8 bit – 256 colors

256 X 256 image with 8 bit color map:
256 X 256 X 8 bits + 256 X 24 = 530.432 Mbits
(66.304 kBytes)

Color-Mapped Images

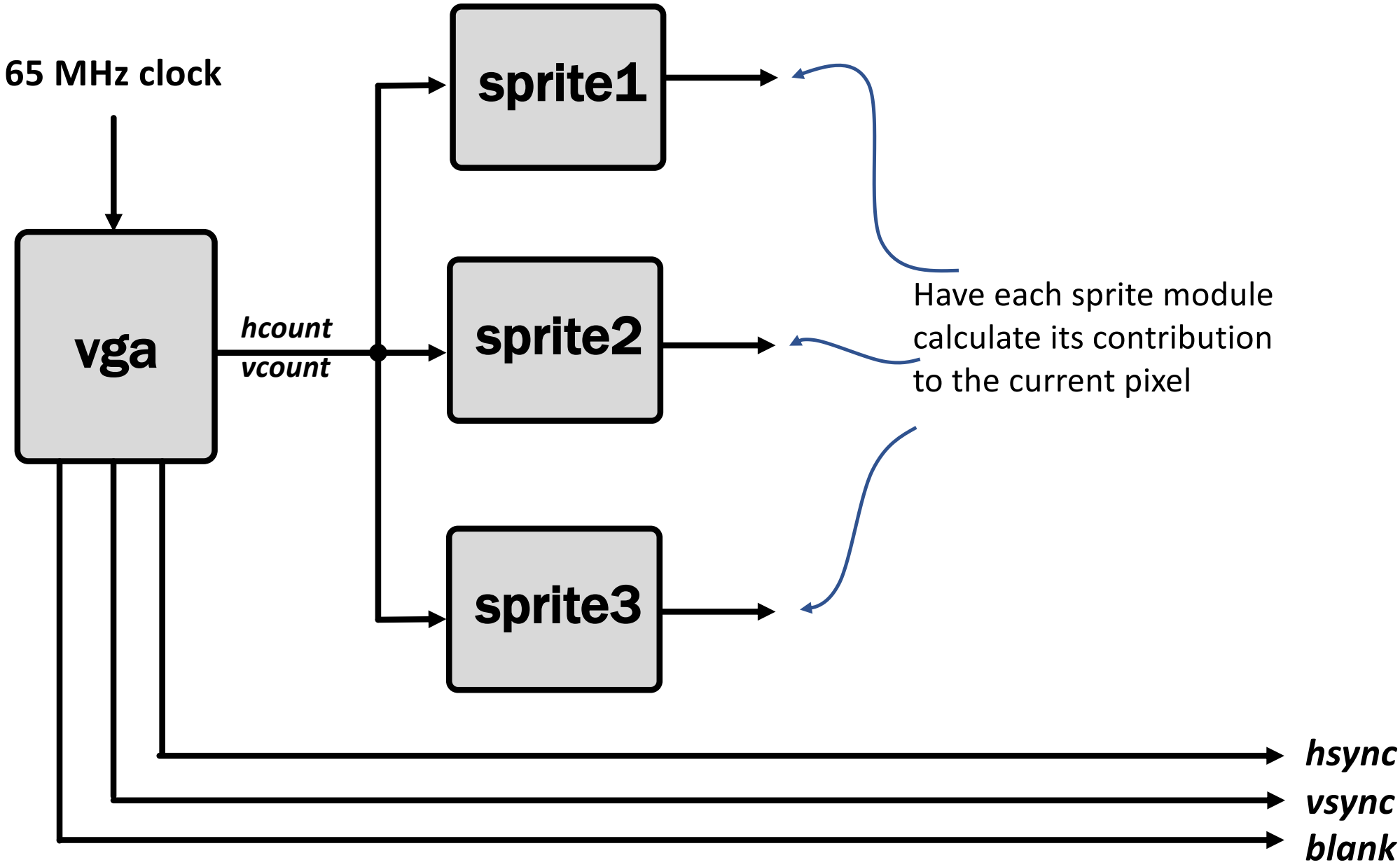
Five VGA Signals!



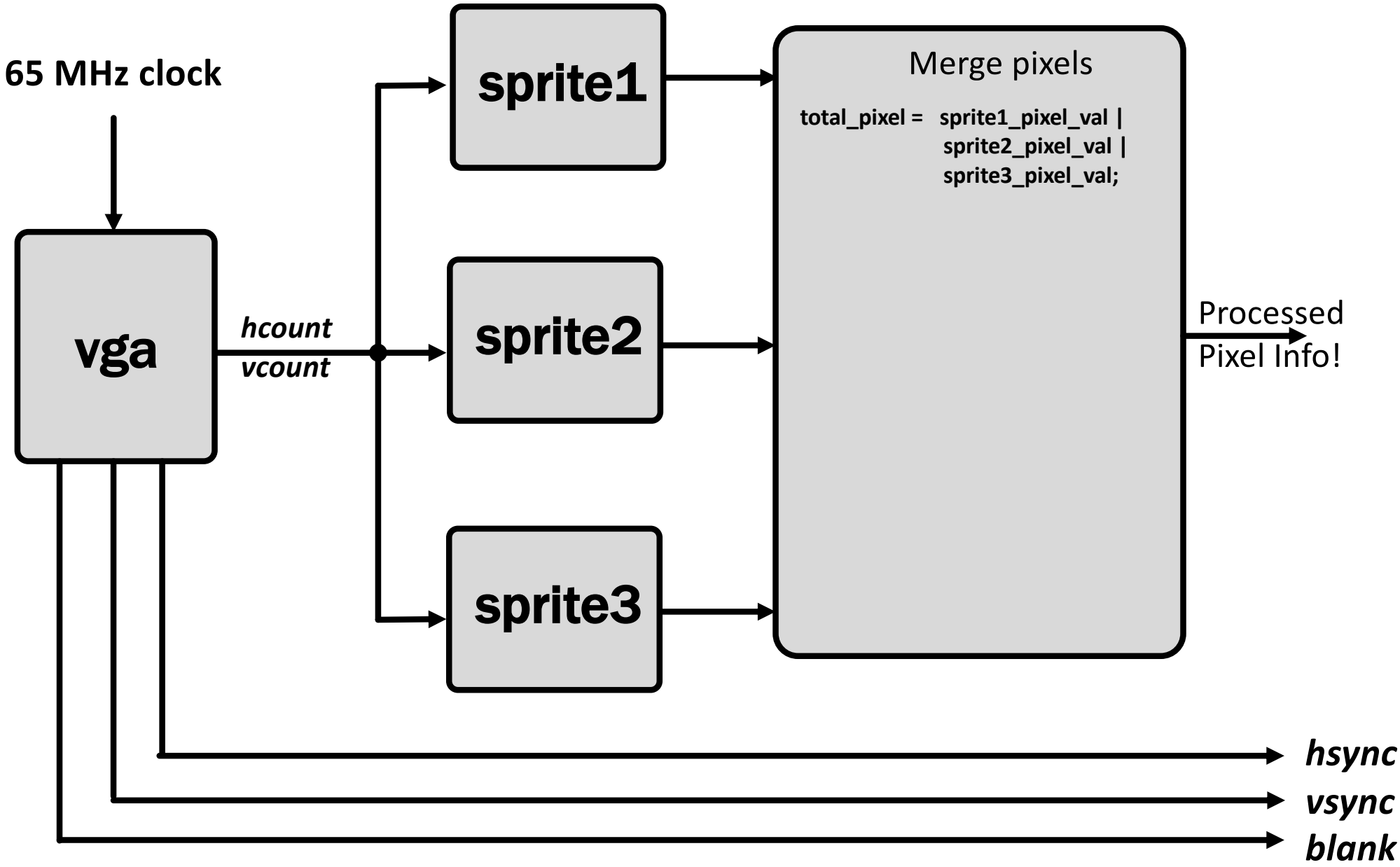
Building a Game?

- Totally possible. How it used to be done
- Once per frame, update your game state!
- Every pixel determine what needs to get drawn!

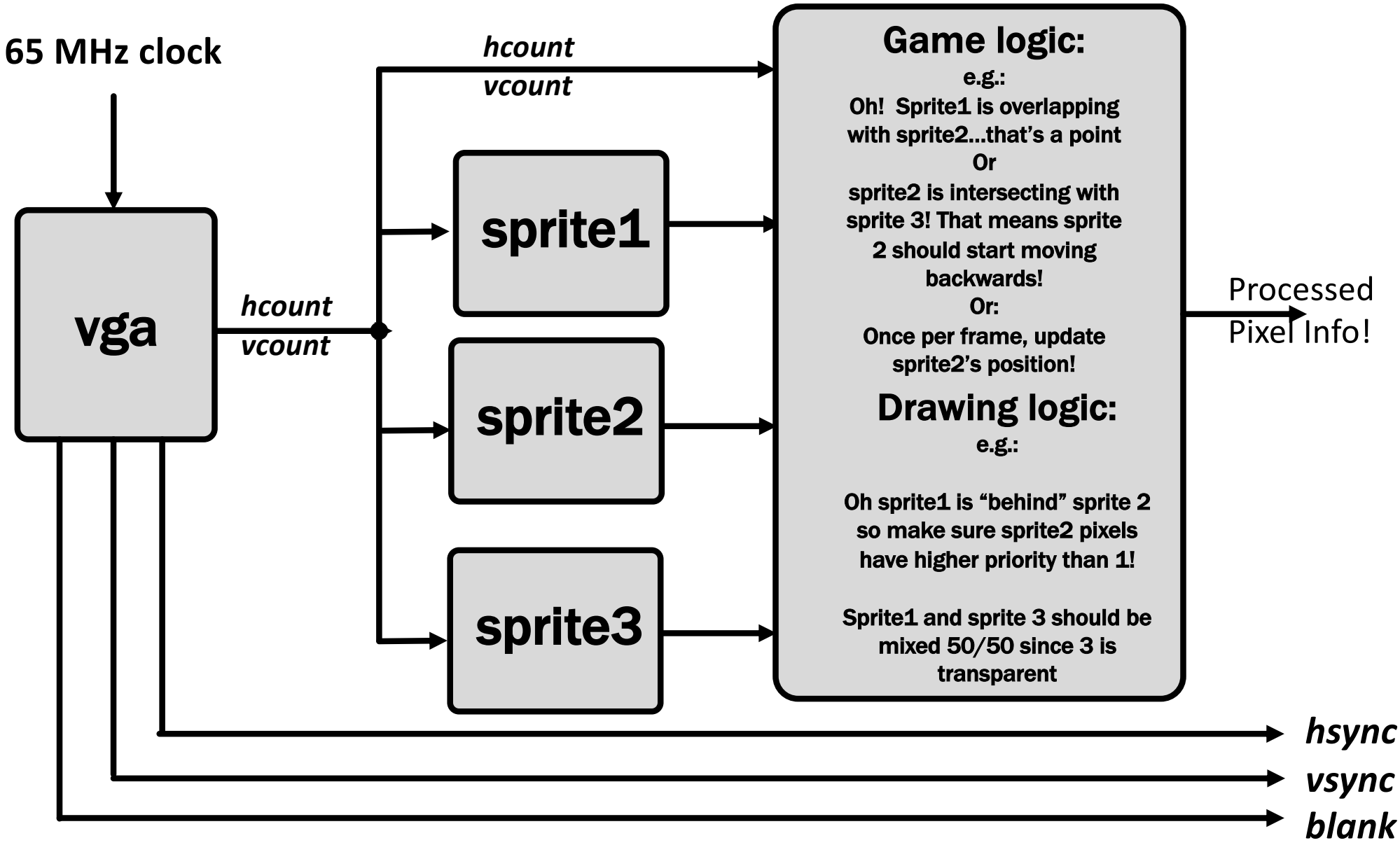
How can a game start to work?



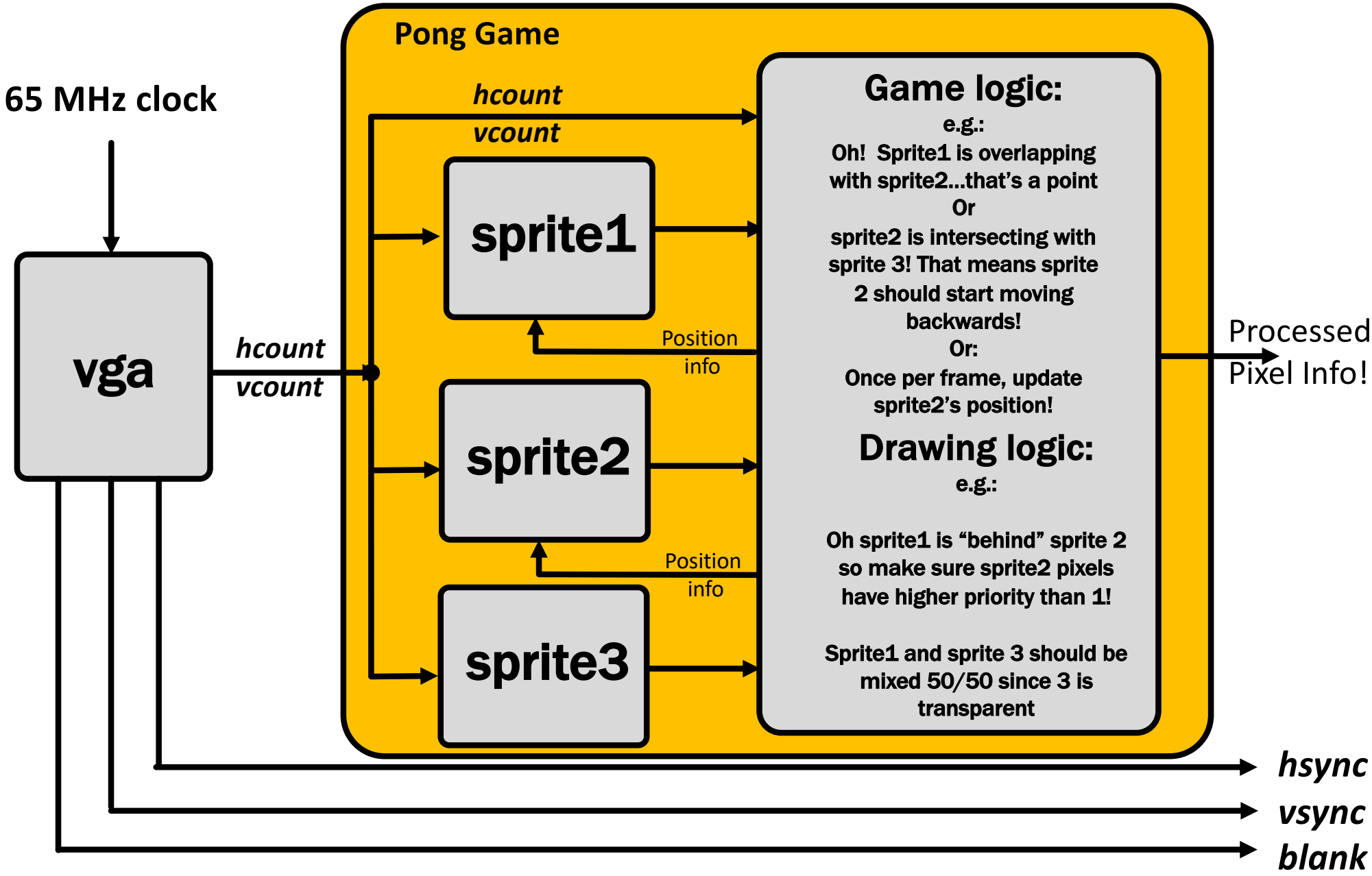
How can a game start to work?



How can a game start to work?



Lab 03!


Pong Game
65 MHz clock
hcount
vcount
sprite1
sprite2
sprite3
vga
hcount
vcount
Position info
Position info
Game logic:
e.g.:
Oh! Sprite1 is overlapping with sprite2...that's a point
Or
sprite2 is intersecting with sprite 3! That means sprite 2 should start moving backwards!
Or:
Once per frame, update sprite2's position!
Drawing logic:
e.g.:
Oh sprite1 is "behind" sprite 2 so make sure sprite2 pixels have higher priority than 1!
Sprite1 and sprite 3 should be mixed 50/50 since 3 is transparent
Processed Pixel Info!
hsync
vsync
blank