Avionics (Command and Data Handling)

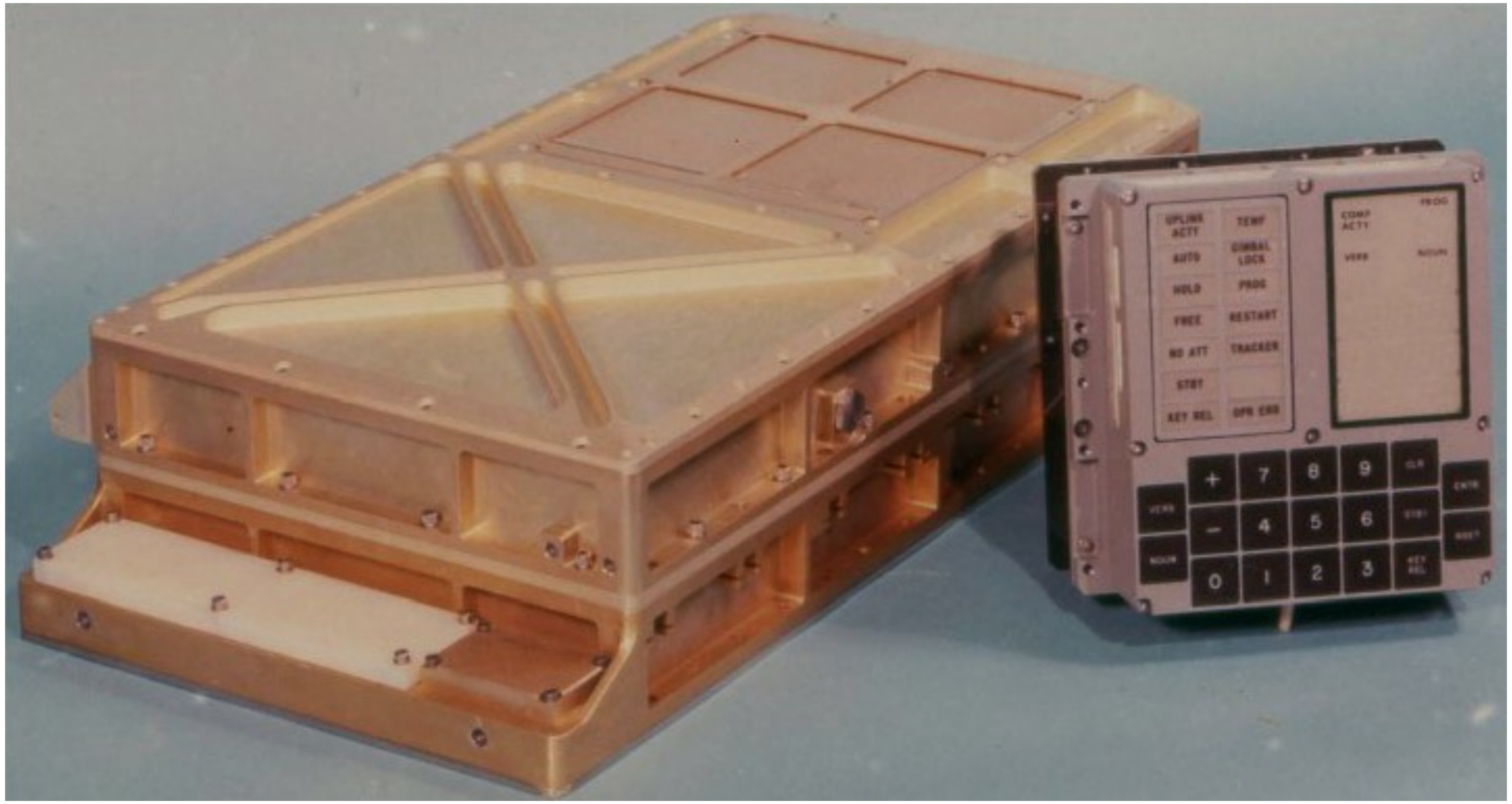
MAE 4160, 4161, 5160

V. Hunter Adams, PhD

Today’s topics:

- Apollo flight computer
- Avionics design process
- Avionics technologies
- Reliability and architecture options
- What could go wrong?

Apollo Guidance Computer (AGC)



The AGC/Display and Keyboard (“DSKY”)

- 16-bit word length (14 bits + sign + parity)
- Memory cycle time: 11.7 microsec
- Add time: 23.4 microsec
- Multiply time: 46.8 microsec
- Divide time: 81.9 microsec
- Memory: 36,864 words (ROM), 2,048 words (RAM)
- 34 normal instructions
- 55 Watts
- 70 lbs

Apollo Guidance Computer (AGC)

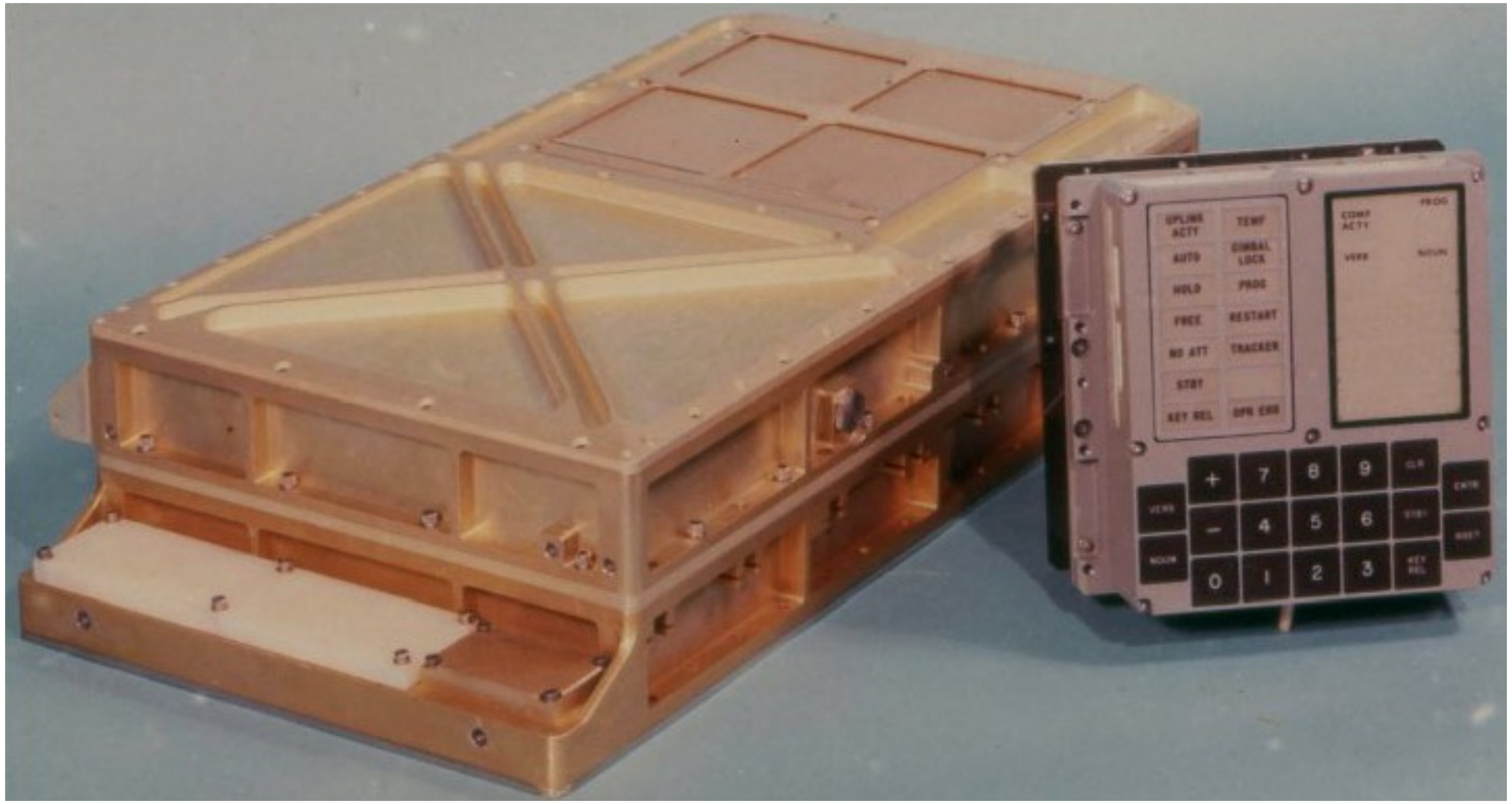
- 16-bit word length (14 bits + sign +

Device	Google Pixel 18W Charger	Huawei 40W SuperCharge	Anker PowerPort Atom PD 2	Apollo 11 Moon Landing Guidance Computer (AGC)
Function	Charges a phone	Charges a phone or maybe a laptop	Charges 2 phones or maybe laptops	• Fly most-of-the-way to moon (CSM) • Land on moon (LEM) • Take off from moon (LEM) • Fly back to Earth (CSM)
Microchip(s)	Weltrend WT6630P	Richtek RT7205	Cypress CYPD4225	Discrete components
Clock Speed	10 MHz	22.7 MHz	48 MHz	1.024 MHz
RAM	512 bytes	"0.75kB"	8KB	2048 15-bit words / 4KB if you include the parity bit in each word
Program Storage Space	8KB	24KB (Mask ROM + OTP)	128KB Flash	36,864 15-bit words / 72KB if you include the parity bit in each word
Instruction Set	Intel 8051 (8-bit)	Unknown	ARM Cortex-M0 32-bit implementing ARMv6-M	16-bit accumulator based
Sources	ChargerLabs Teardown WT6630P Datasheet	ChargerLabs Teardown RT2705 Datasheet	ChargerLabs Teardown CYPD4225 Datasheet	CPU description Memory Functional overview

- 55 Watts
- 70 lbs

The AGC/Display and Keyboard ("DSKY")

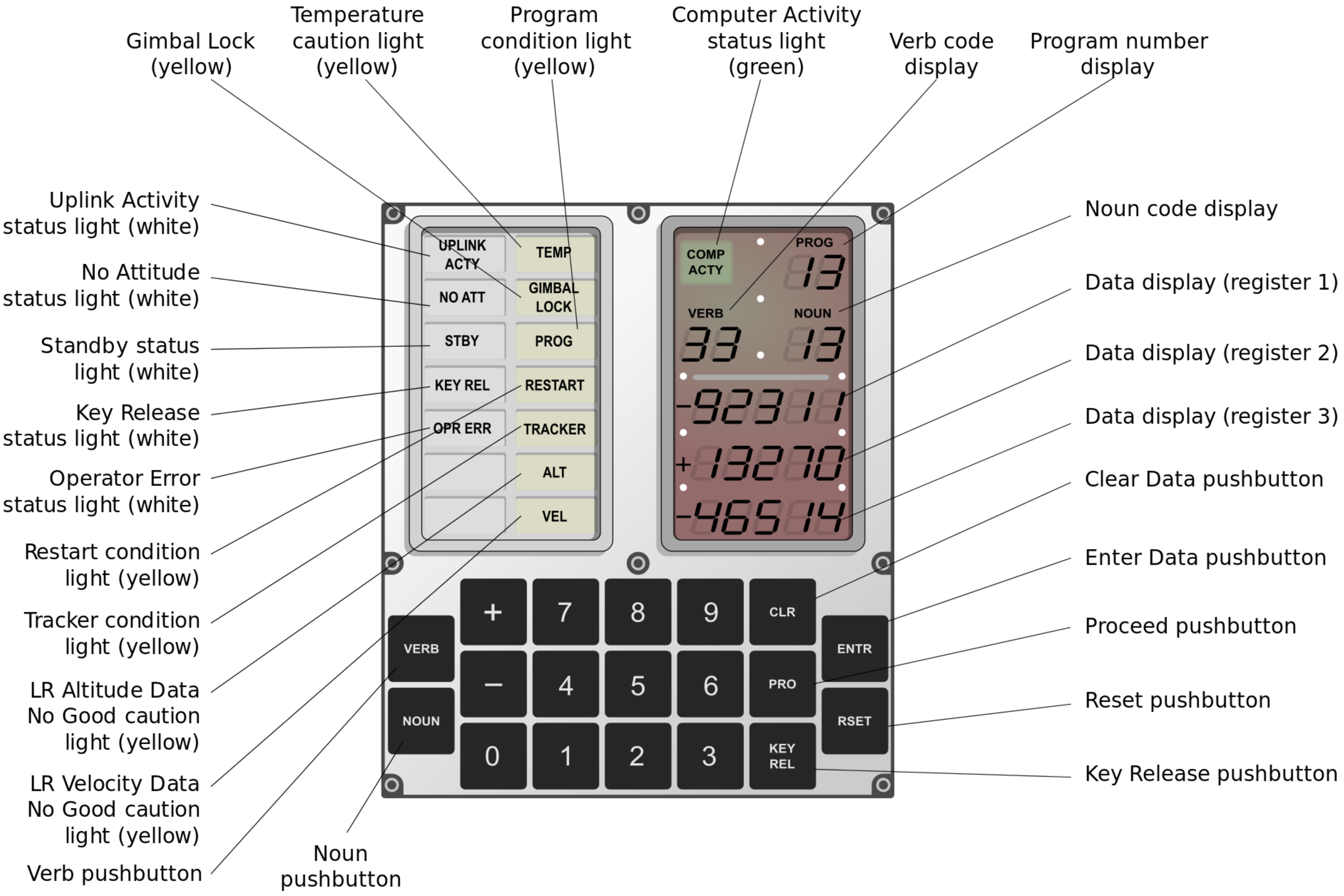
Apollo Guidance Computer (AGC)



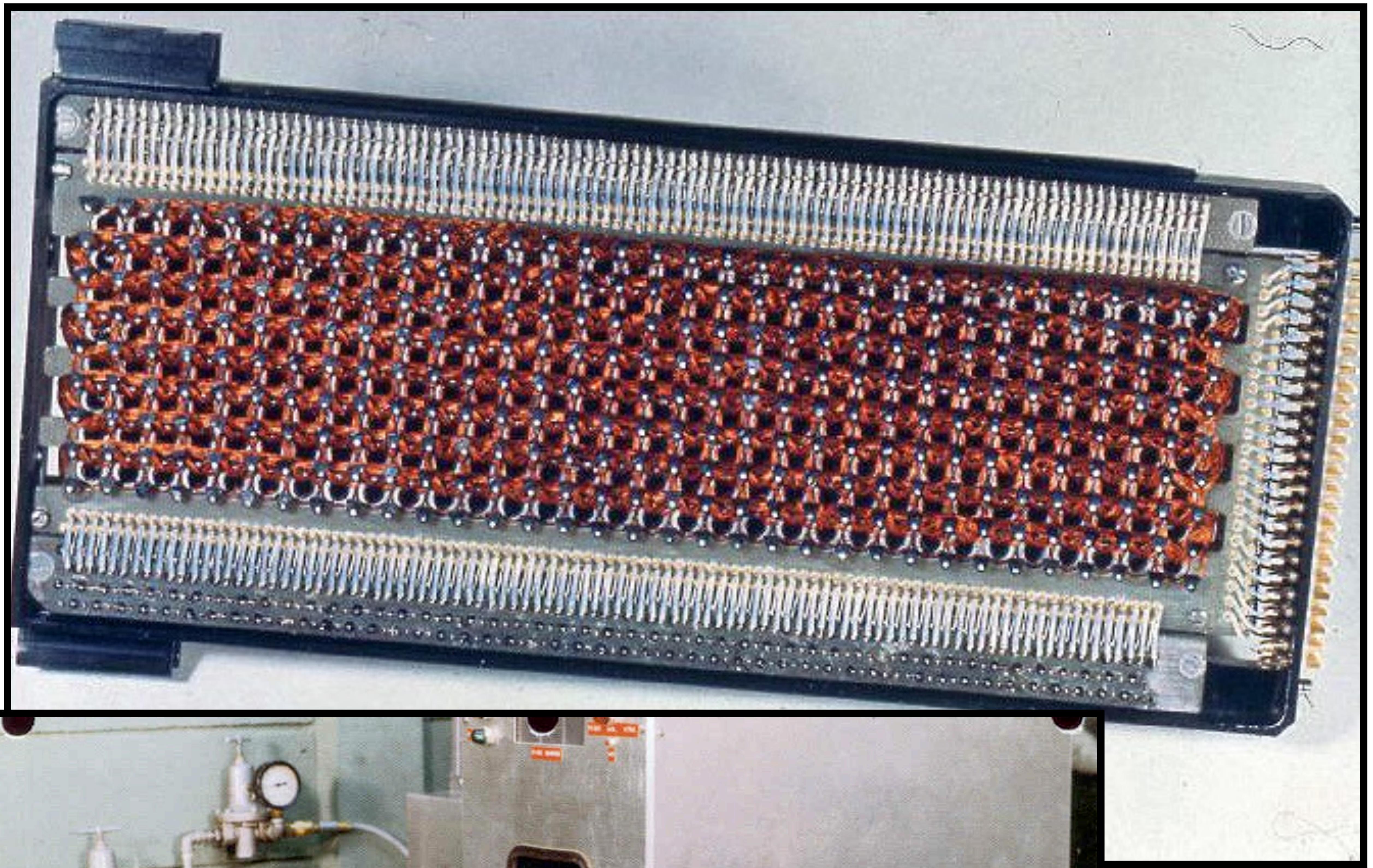
The AGC/Display and Keyboard ("DSKY")

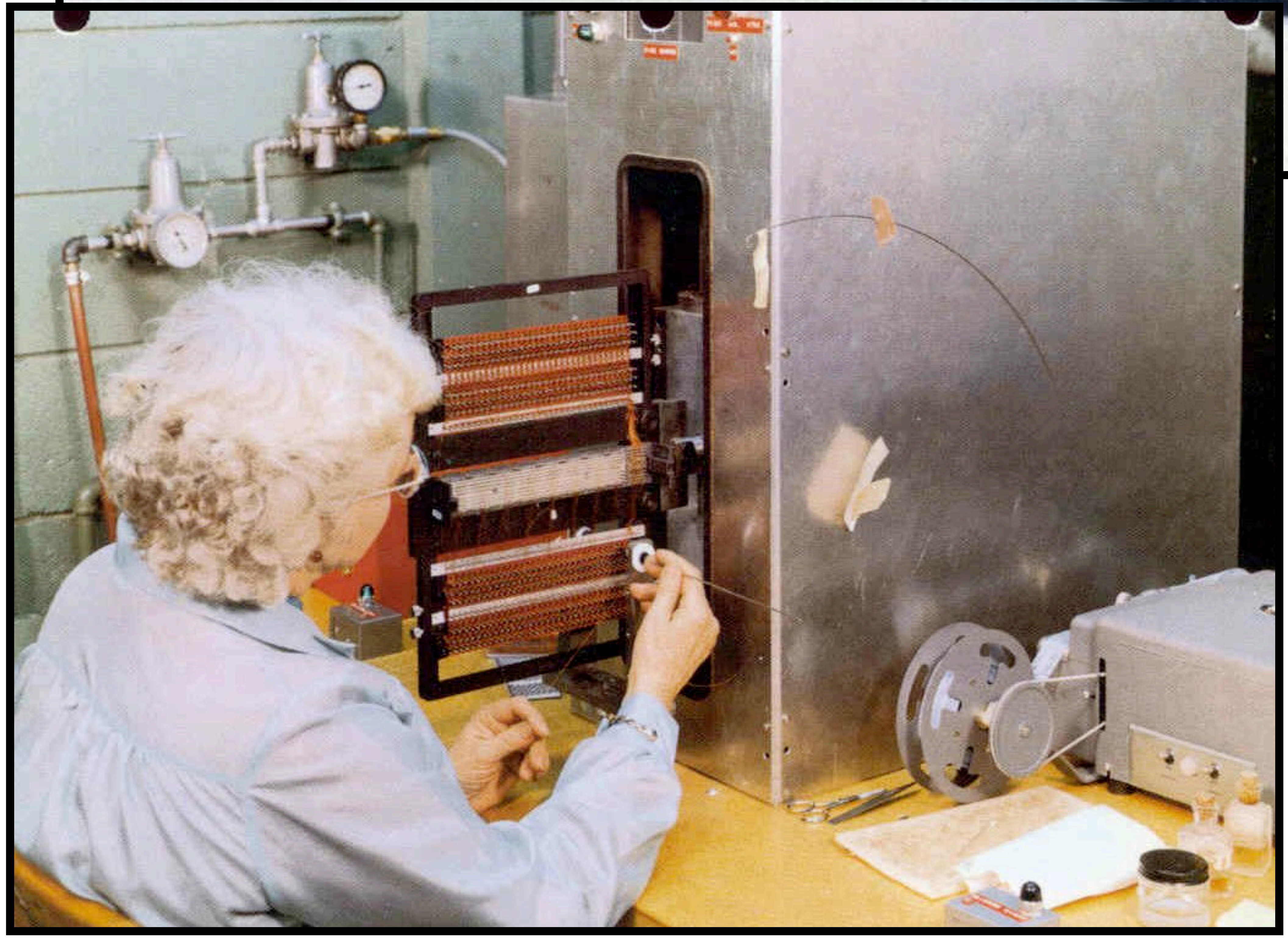
- 16-bit word length (14 bits + sign + parity)
- Memory cycle time: 11.7 microsec
- Add time: 23.4 microsec
- Multiply time: 46.8 microsec
- Divide time: 81.9 microsec
- Memory: 36,864 words (ROM), 2,048 words (RAM)
- 34 normal instructions
- 55 Watts
- 70 lbs

Why do we need recursive estimators like Kalman filters?



Gimbal Lock (yellow)
Temperature caution light (yellow)
Program condition light (yellow)
Computer Activity status light (green)
Verb code display
Program number display
Uplink Activity status light (white)
No Attitude status light (white)
Standby status light (white)
Key Release status light (white)
Operator Error status light (white)
Restart condition light (yellow)
Tracker condition light (yellow)
LR Altitude Data No Good caution light (yellow)
LR Velocity Data No Good caution light (yellow)
Verb pushbutton
Noun pushbutton
Noun code display
Data display (register 1)
Data display (register 2)
Data display (register 3)
Clear Data pushbutton
Enter Data pushbutton
Proceed pushbutton
Reset pushbutton
Key Release pushbutton
UPLINK ACTY
TEMP
NO ATT
GIMBAL LOCK
STBY
PROG
KEY REL
RESTART
OPR ERR
TRACKER
ALT
VEL
COMP ACTY
PROG
13
VERB
NOUN
33
13
-92311
+13270
-46514
+
7
8
9
CLR
VERB
ENTR
−
4
5
6
PRO
NOUN
RSET
0
1
2
3
KEY REL

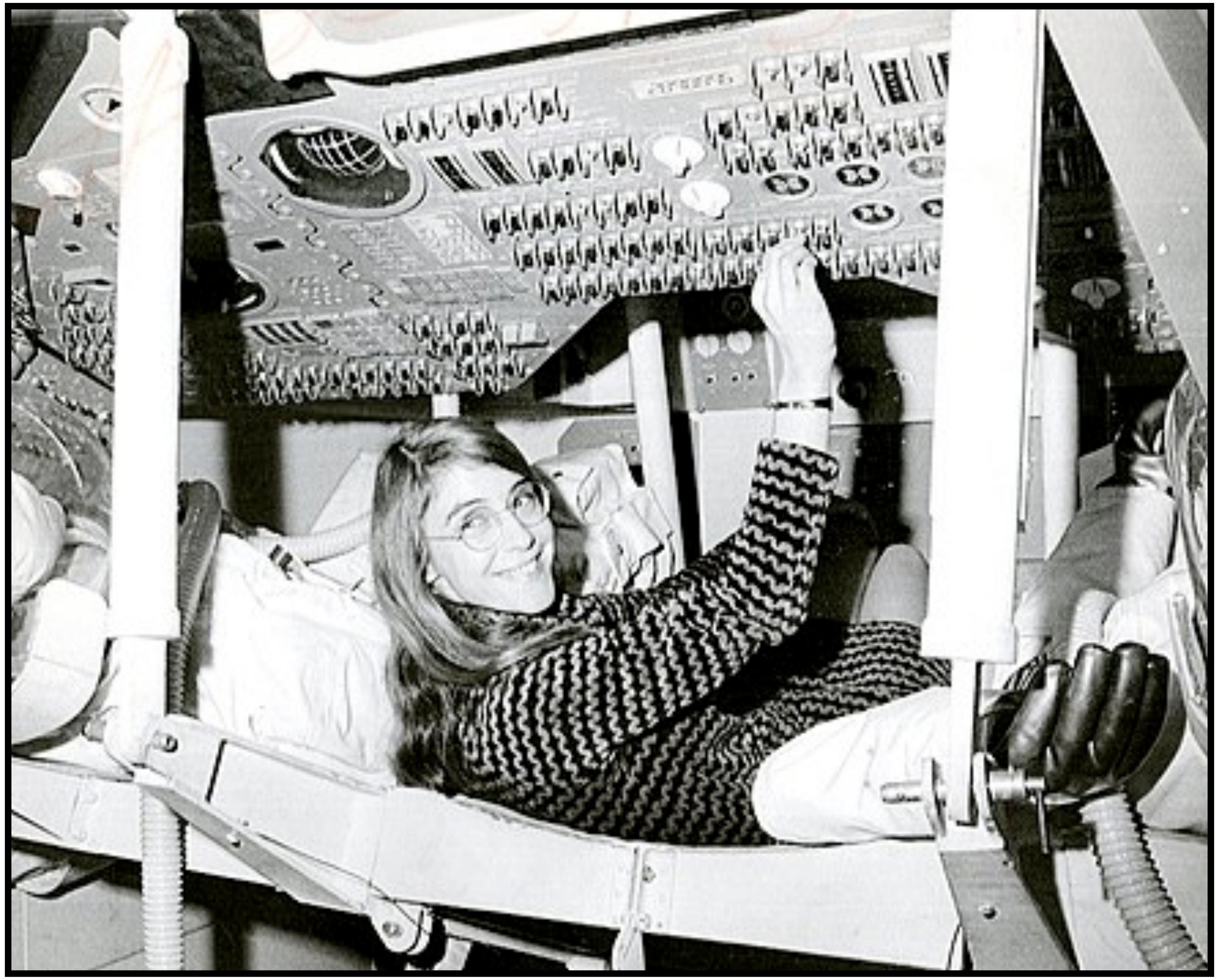




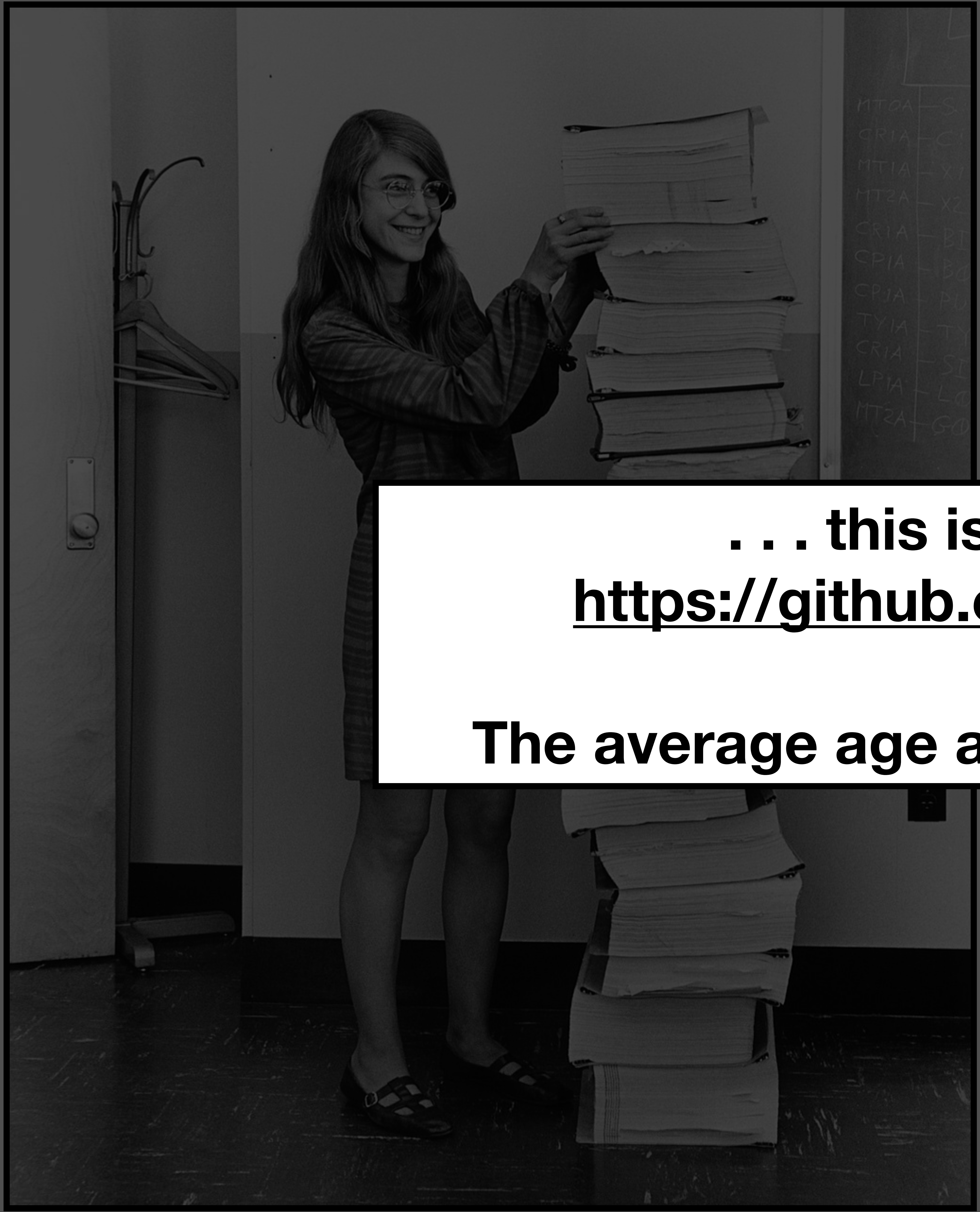
- Hand-woven rope-core memory (wires woven around magnetic cores)
- ~72 kilobytes per cubic foot

Hamilton next to her software

- Programmed in assembly language
- Most of the software was stored on read-only core rope memory, but some was stored on read-write magnetic-core memory and could be overwritten via the DSKY
- Ran a simple real-time operating system for scheduling tasks
- Capable of double-precision trigonometric, scalar, and vector arithmetic
- Software was implemented by a team run by **Margaret Hamilton**. Software development comprised 1400 person-years of effort, with a peak workforce of 350 people
- Hamilton won the Presidential Medal of Freedom in 2016 for her efforts



- Programmed in assembly language
- Most of the software was stored on read-only core rope memory, but some was stored on read-write magnetic-core memory and could be overwritten via the DSKY
- Ran a simple real-time operating system for scheduling tasks
- Capable of double-precision trigonometric, scalar, and vector arithmetic
- Software was implemented by a team run by **Margaret Hamilton**. Software development comprised 1400 person-years of effort, with a peak workforce of 350 people
- Hamilton won the Presidential Medal of Freedom in 2016 for her efforts



Hamilton next to her software

- Programmed in assembly language
- Most of the software was stored on read-only core rope memory, but some was stored on read-write magnetic-core memory and could be overwritten via the DSKY
- ating

ctor

. . . this is now open-sourced.
https://github.com/chrislgarry/Apollo-11

The average age at NASA was 28 during Apollo.

- Software was implemented by a team run by **Margaret Hamilton**. Software development comprised 1400 person-years of effort, with a peak workforce of 350 people
- Hamilton won the Presidential Medal of Freedom in 2016 for her efforts

BURN_BABY_BURN - - MASTER_IGNITION_ROUTINE.agc

```
# Copyright:    Public domain.
# Filename:     BURN_BABY_BURN--MASTER_IGNITION_ROUTINE.agc
# Purpose:      Part of the source code for Luminary 1A build 099.
#               It is part of the source code for the Lunar Module's (LM)
#               Apollo Guidance Computer (AGC), for Apollo 11.
# Assembler:    yaYUL
# Contact:      Ron Burkey <info@sandroid.org>.
# Website:      www.ibiblio.org/apollo.
# Pages:        731-751
# Mod history:  2009-05-19 RSB  Adapted from the corresponding
#                               Luminary131 file, using page
#                               images from Luminary 1A.
#               2009-06-07 RSB  Corrected 3 typos.
#               2009-07-23 RSB  Added Onno's notes on the naming
#                               of this function, which he got from
#                               Don Eyles.
#
```

Hamilton next to her software

bly language

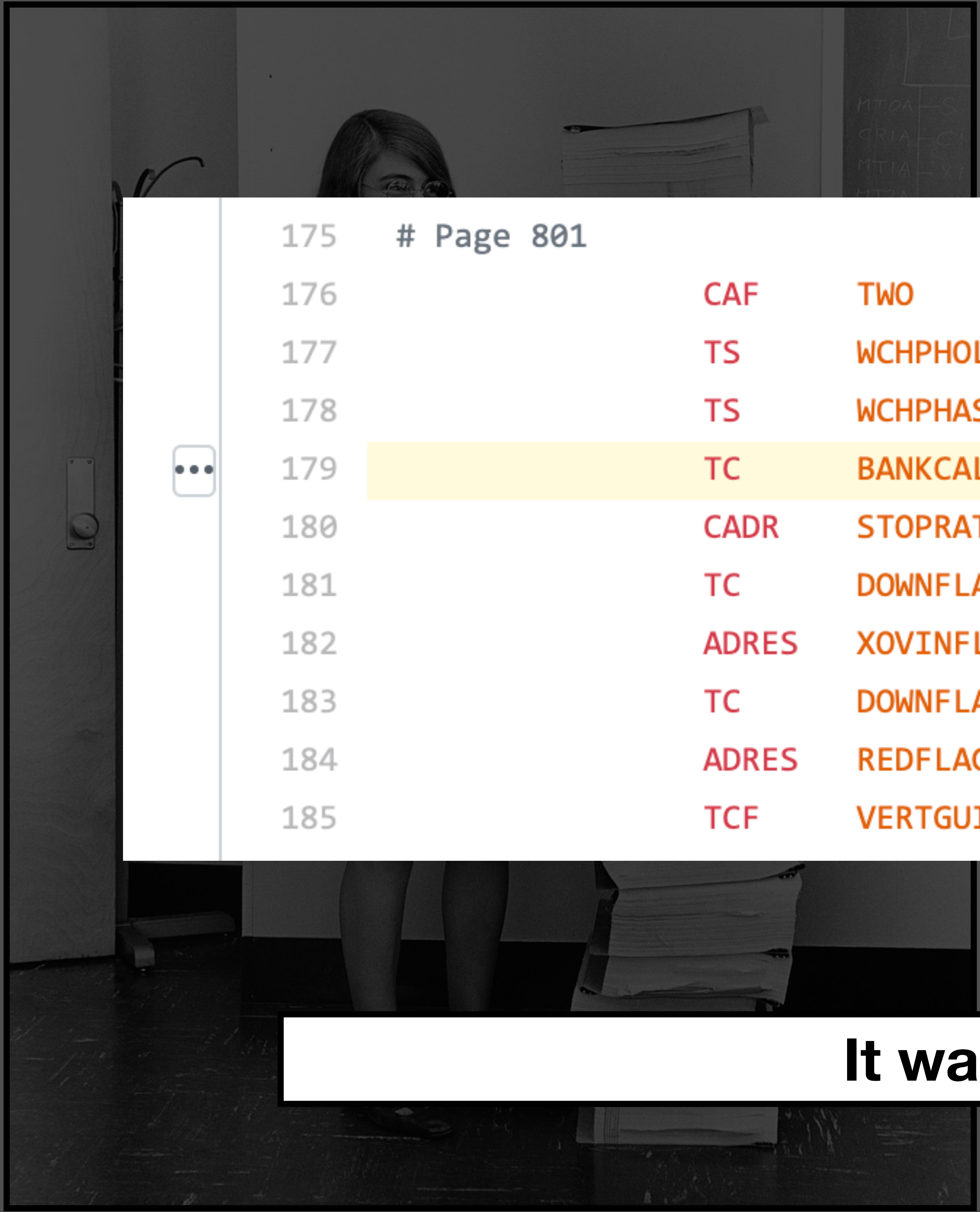
as stored on
emory, but
ad-write
and could be
KY

operating
tasks

cision
nd vector

nted by a
Hamilton.
Software development comprised
1400 person-years of effort, with a

- Hamilton won the Presidential Medal of Freedom in 2016 for her efforts



Hamilton next to her software

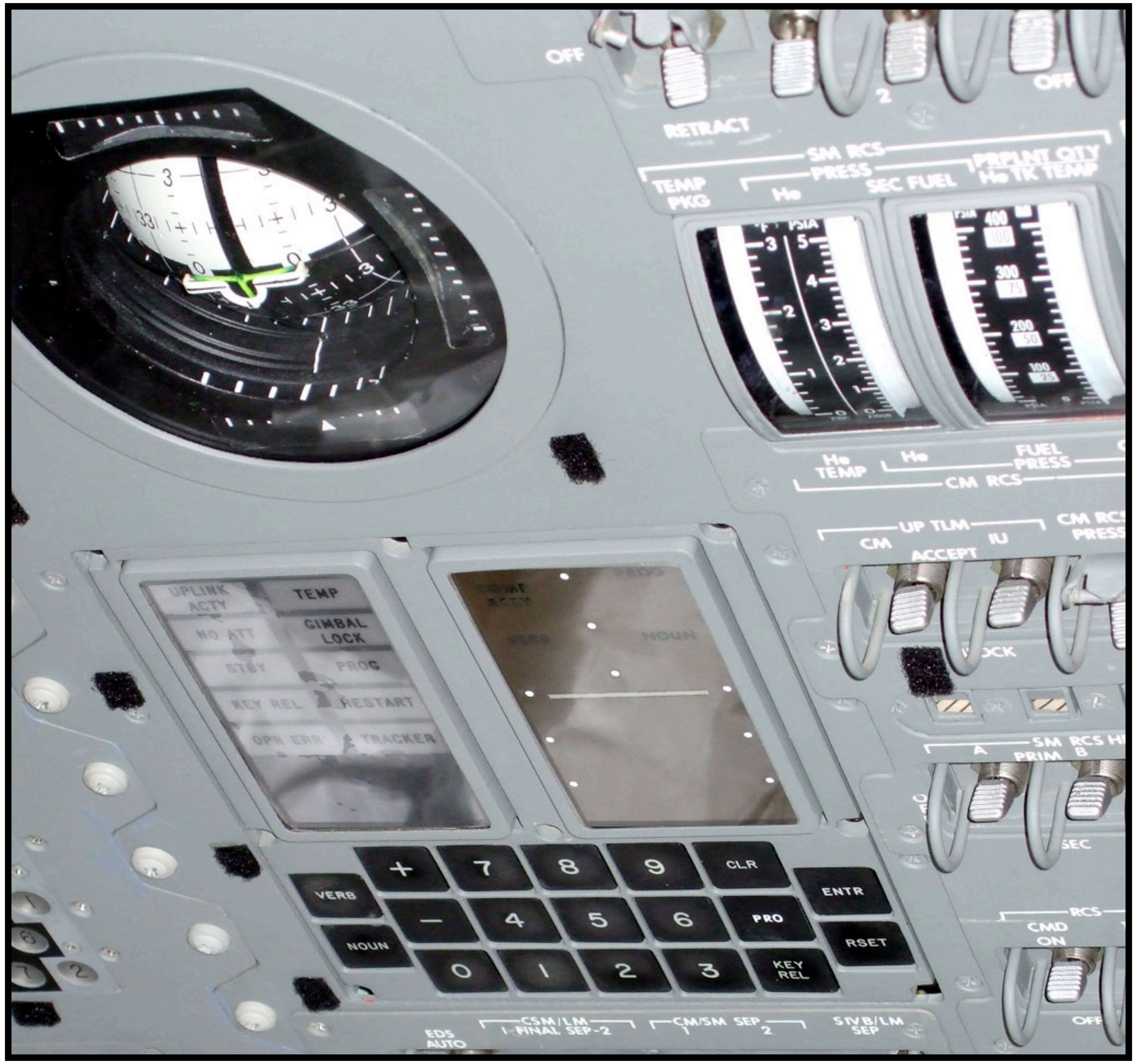
```
# Page 801
            CAF     TWO             # WCHPHASE = 2 ---> VERTICAL: P65,P66,P67
            TS      WCHPHOLD
            TS      WCHPHASE
            TC      BANKCALL        # TEMPORARY, I HOPE HOPE HOPE
            CADR    STOPRATE        # TEMPORARY, I HOPE HOPE HOPE
            TC      DOWNFLAG        # PERMIT X-AXIS OVERRIDE
            ADRES   XOVINFLG
            TC      DOWNFLAG
            ADRES   REDFLAG
            TCF     VERTGUID
```

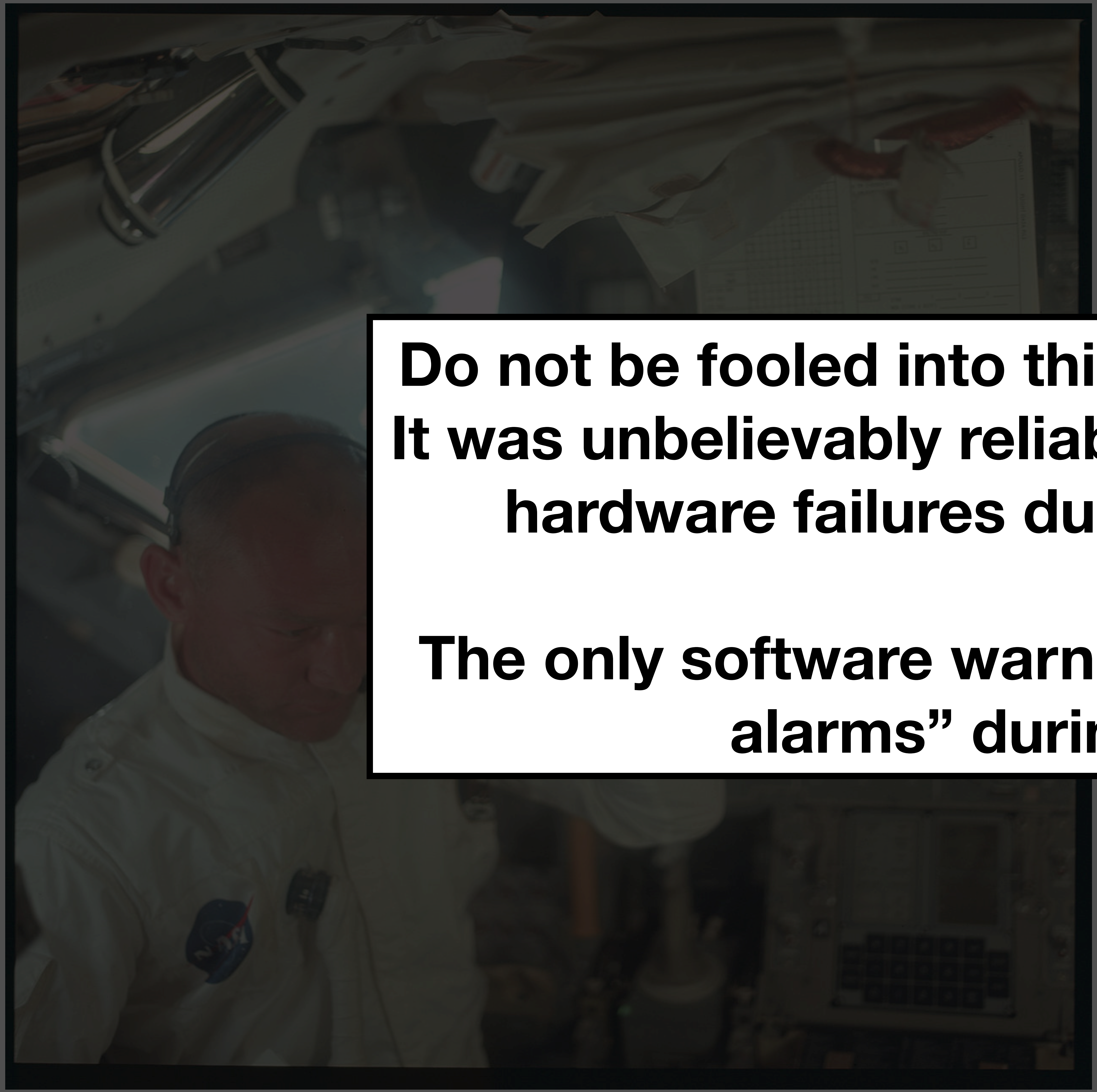
It was not temporary.

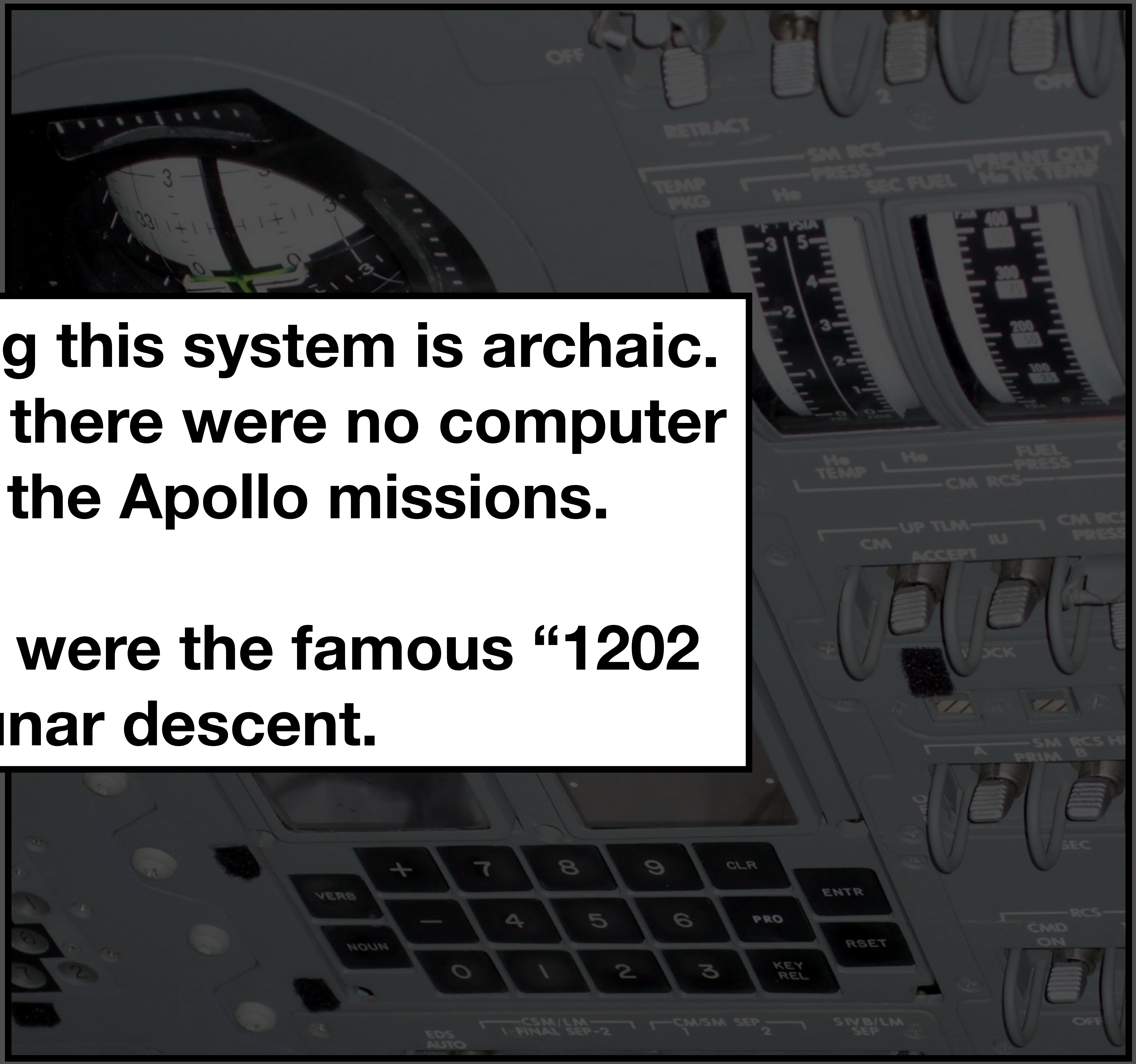
- Programmed in assembly language
- Most of the software was stored on

team run by **Margaret Hamilton**. Software development comprised 1400 person-years of effort, with a

- Hamilton won the Presidential Medal of Freedom in 2016 for her efforts


OFF
RETRACT
SM RCS
TEMP PKG
PRESS
He
SEC FUEL
PRPLNT QTY
He TK TEMP
PSIA
He TEMP
He
FUEL PRESS
CM RCS
UP TLM
CM
ACCEPT
IU
CM RCS PRESS
UPLINK ACTY
TEMP
NO ATT
GIMBAL LOCK
STBY
PROG
KEY REL
RESTART
OPR ERR
TRACKER
VERB
NOUN
CLR
ENTR
PRO
RSET
KEY REL
SM RCS
PRIM
SEC
RCS
CMD ON
EDS AUTO
CSM/LM
FINAL SEP
CM/SM SEP
S IVB/LM SEP





Do not be fooled into thinking this system is archaic. It was unbelievably reliable - there were no computer hardware failures during the Apollo missions.

The only software warnings were the famous “1202 alarms” during lunar descent.

1202 Alarms

- The flight computer generated unanticipated warnings during Apollo 11’s lunar descent
- During descent, the lander turned on their rendezvous radar to track the command module as a safety measure
- The radar measurements caused frequent interrupts in the Apollo Guidance Computer, preventing spurious threads from terminating
- When a new task was sent to the computer, there was no memory left for it to go - 1201/1202 alarms
- Computer autonomously rebooted, but the problem persisted. Aldrin noticed that the alarm seemed to be correlated with the times that he displayed the lander’s velocity. This extra task pushed the memory over the edge and caused a 1202.
- Because these reboots occurred a few minutes apart, no navigation data was lost in the reboots and Houston gave Apollo 11 a “GO,” in spite of the alarms. This failsafe software saved the mission.

Today's topics:

- Apollo flight computer
- **Avionics design process**
- Avionics technologies
- Reliability and architecture options
- What could go wrong?

C&DH Design Process

1. Allocate mission and system requirements
2. Define the computer system’s operational modes and states
3. Functionally partition and allocate the computational requirements
4. Evaluate internal and external interfaces
5. Select baseline architecture
6. Form the baseline system specification

Let’s look at each step individually.

C&DH Design Process

1. Allocate mission and system requirements

Table 20-4. Design Drivers for Computer Systems. These are factors that we evaluate throughout the design process. When flowing down mission requirements, including system level processing requirements, we must be careful to design hardware and software with the "ilities" in the fourth column in mind.

Mission Requirements	System Level Processing Requirements	Computer Level Requirements	Additional Requirements "ilities"
• Customer Needs • Expected Availability – Weeks – Months – Year or More • Number of Satellites • Number and Location of Ground Stations • Level of Autonomy • Security Requirements • Programmatic Issues – Cost – Schedule – Risk	• Functional Capabilities • Processing Partitioning – Payload vs. Spacecraft – Onboard vs. Ground • Physical Characteristics – Size – Weight – Power – Radiation • Communication Protocol – Commercial Digital Standards – Commercial Analog Standards – Protection / Encryption	• Throughput • Memory • Radiation Hardness • Development Tools • COTS Software availability • Emulator / Engineering Model availability	• Testability • Feasibility • Usability • Reusability • Reliability • Flexibility • Maintainability • Interchangeability • Replaceability

C&DH Design Process

1. Allocate mission and system requirements

Main Requirements:

- Throughput (instructions per second)
- Data storage
    - Firmware (ROM) - kb
    - O/S data (RAM) - Mb
    - Data storage (Disk) - Gb
- Radiation hardness (10 krad LEO, ~Mrad interplanetary)
- Reliability/fault tolerance
- Flexibility: change after launch

Main Functions:

- Running flight software
- Executing commands
- Storing data
- Processing data
- Distributing data

C&DH Design Process

1. Allocate mission and system requirements

Main Requirements:

- Throughput (instructions per second)
- Data storage
  - O/S data (RAM) - Mb
  - Data storage (Disk) - Gb
- Radiation hardness (10 krad LEO, ~Mrad interplanetary)
- Reliability/fault tolerance
- Flexibility: change after launch

How do we estimate these requirements?

Main Functions:

- Running flight software
- Executing commands
- Processing data
- Distributing data

C&DH Design Process

1. Allocate mission and system requirements

1. List all **applications and functions** allocated to the computer
2. Estimate the memory space needed for each application/function (either by analogy or bottom-up estimate)
3. Estimate throughput
    1. Determine the frequency of the function (executions/second)
    2. Estimate instructions per execution and cycles per instruction
4. List all utility functions, determine operating system requirements
    1. Determine requirements for concurrent processes, interrupts, realtime tasks
5. Determine margins for growth/spare capacity

C&DH Design Process

1. Allocate mission and system requirements

1. List all **applications and functions** allocated to the computer

- **Payload**: pointing, on/off
- **T&C**: telemetry and command processing
- **Attitude/orbit sensor processing**: gyros, star trackers, sun sensors, etc.
- **ADCS algorithms**: Kalman filters, orbit propagation, integration, etc.
- **Attitude control processing**: thrusters, reaction wheels, torque coils, etc.
- **Fault detection**: monitoring, identification, and correction
- **Power management**: battery charging, solar array pointing
- **Thermal management**: heaters, louvers, coolers, pointing
- **Momentum management**: momentum wheels
- **Utilities**: basic math functions, matrix algebra, time management, rotations

C&DH Design Process

1. Allocate mission and system requirements

1. List all **applications and functions** allocated to the computer

Table 20-2. Definitions Associated with Computer Systems. Often when discussing computer system design and development we use terms which have a specific meaning to those involved in the discipline.

Embedded Systems	A built-in processor or microprocessor, providing real-time control as a component of a larger system, often with no direct user interface.
Real-Time Processing	Handling or processing information at the time events occur or when the information is first created. Typically, embedded or on board processing is real-time.
Hard Real-Time	Requiring precise timing to achieve their results, where missing the time boundary has severe consequences. Examples include attitude control software and telemetry downlink. [Stankovic and Ramamritham, 1988].
Soft Real-Time	Requiring only that the tasks are performed in a timely manner, the consequences of missing a time boundary are often degraded, but continuous, performance. Examples include orbit control software and general status or housekeeping.
Operating System Software	Manages the computer's resources such as input/output devices, memory, and scheduling of application software.
Application Flight Software (FSW)	Mission specific software which does work required by the user or the mission rather than in support of the computer.

C&DH Design Process

1. Allocate mission and system requirements

1. List all **applications and functions** allocated to the computer
2. Estimate the memory space needed for each application/function (either by analogy or bottom-up estimate)
3. Estimate throughput
    1. Determine the frequency of the function (executions/second)
    2. Estimate instructions per execution and cycles per instruction
4. List all utility functions, determine operating system requirements
    1. Determine requirements for concurrent processes, interrupts, realtime tasks
5. Determine margins for growth/spare capacity

C&DH Design Process

1. Allocate mission and system requirements

Conceptual example of throughput estimation

Function	Execution frequency (Hz)	Instructions per exec	Instructions per sec	Cycles per instruction	Cycles per second
Read battery sensor temperature	4	1	4	1	4
Noise filter	300	100	30 kIPS	5	150k
Convert sensor coordinates to S/C coordinates	300	50	15kIPS	5	75k
Propagate orbit	200	20,000	4 MIPS	5	20M

Total: 20.25 MHz

C&DH Design Process

1. Allocate mission and system requirements

1. List all **applications and functions** allocated to the computer
2. Estimate the memory space needed for each application/function (either by analogy or bottom-up estimate)
3. Estimate throughput
    1. Determine the frequency of the function (executions/second)
    2. Estimate instructions per execution and cycles per instruction
4. List all utility functions, determine operating system requirements
    1. Determine requirements for concurrent processes, interrupts, realtime tasks
5. Determine margins for growth/spare capacity

C&DH Design Process

1. Allocate mission and system requirements
2. Define the computer system's operational modes and states
3. Functionally partition and allocate the computational requirements
4. Evaluate internal and external interfaces
5. Select baseline architecture
6. Form the baseline system specification

Let's look at each step individually.

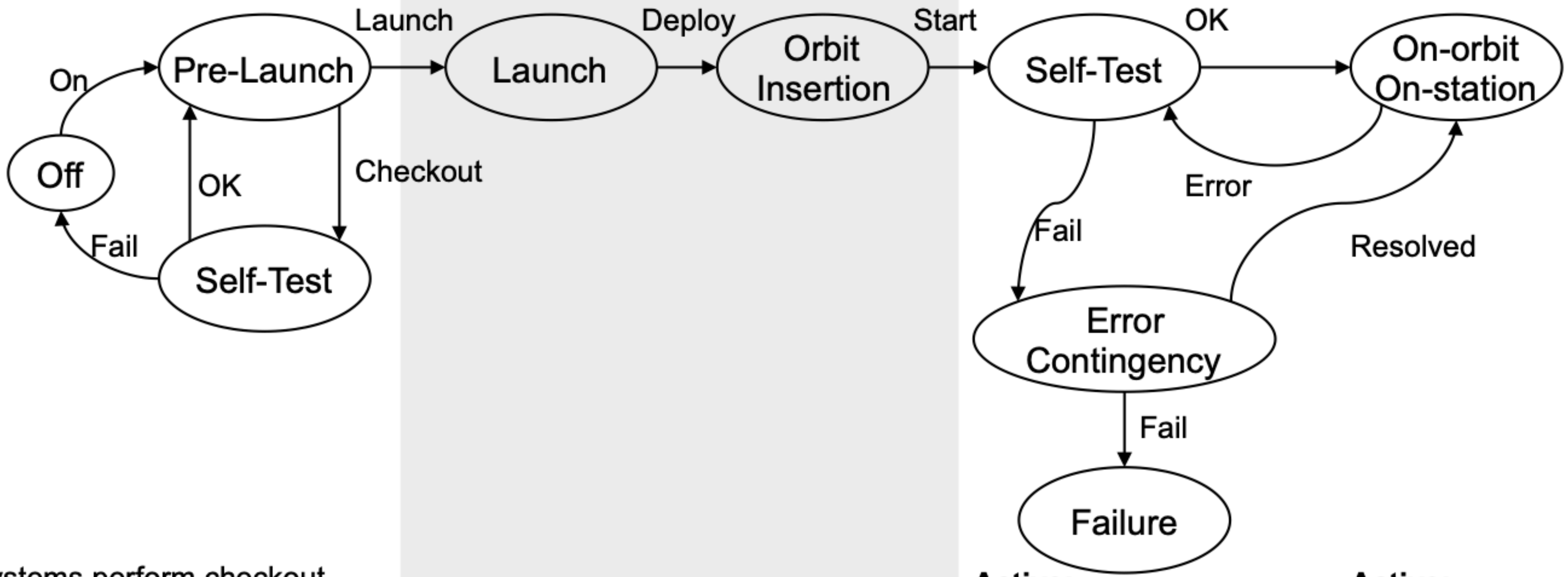
C&DH Design Process

2. Define the computer system’s operational modes and states

1. Develop a state diagram consistent with functional requirements
2. Model different operational stages as different states
3. Ensure degradation/failure states are modeled
4. Consider the effects on ground/ops for all states.

C&DH Design Process

2. Define the computer system's operational modes and states



-All subsystems perform checkout prior to launch

-Avionics active

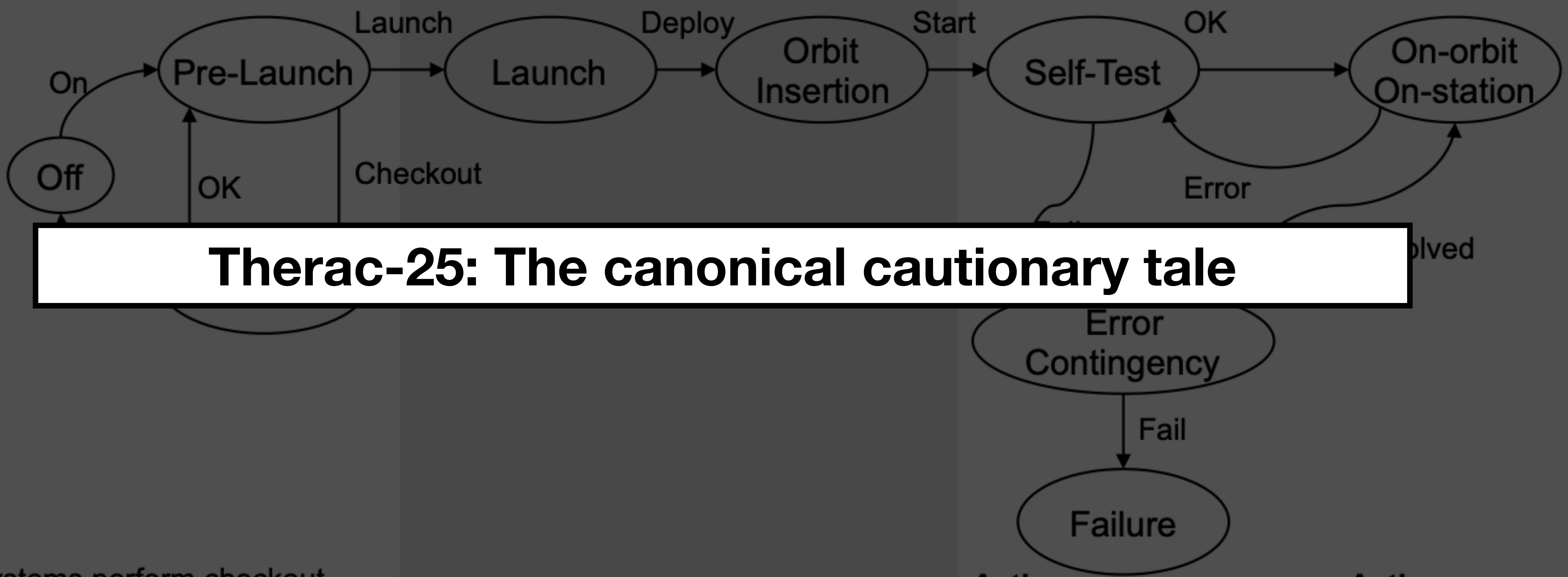
-Bus avionics do not play a relevant role during these stages
-Communication of telemetry if possible

Active:
-Power
-Avionics
  -ACS if possible
-Comm
-Subsystem under test

Active:
-All

C&DH Design Process

2. Define the computer system's operational modes and states



Therac-25: The canonical cautionary tale

-All subsystems perform checkout prior to launch

-Avionics active

-Bus avionics do not play a relevant role during these stages
-Communication of telemetry if possible

Active:
-Power
-Avionics
-ACS if possible
-Comm
-Subsystem under test

Active:
-All

Therac-25

A series of accidents highlighted the dangers of software control of safety-critical systems, with lessons learned that extend to spacecraft.

- Therac-25 was a radiation therapy machine from 1982
- Two modes: **electron-beam** (5-25 MeV) and **Megavolt X-ray** (25 MeV)
- Megavolt X-Ray mode involved sending a 100x-higher current beam of electrons through a target, which interacted with the beam to produce X-rays which were delivered to the patient
- There were no hardware/software interlocks. A technician could select X-ray mode without having the target in place, delivering lethal doses of radiation to patients
- 6 people were overdosed
- The consequence of poor software design and development practice, not the consequence of bad code

Therac-25

Therac-25

A series of accidents highlighted the dangers of software control of safety-critical systems, with lessons learned that extend to spacecraft.

- Therac-25 was a radiation therapy machine from 1982
- Two modes: **electron-beam** (5-25 MeV) and **Megavolt X-ray** (25 MeV)
- Megavolt X-Ray mode current beam of electrons through a target, which interacted with the beam to produce X-rays which were delivered to the patient
- There were no hardware/software interlocks. A technician could select X-ray mode without having the target in place, delivering lethal doses of radiation to patients
- 6 people were overdosed
- The consequence of poor software design and development practice, not the consequence of bad code

Inadequate analysis of failure modes.

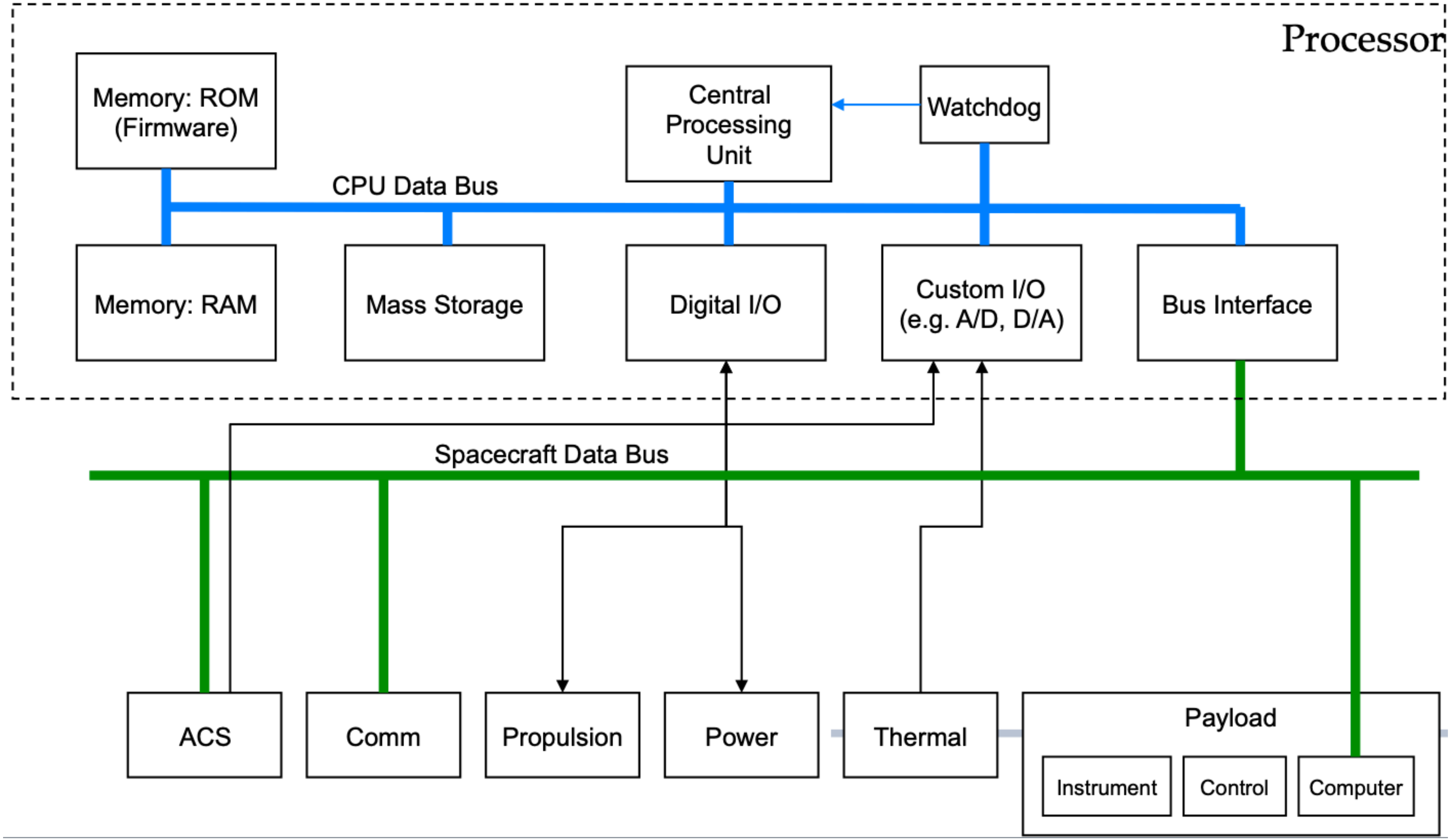
Therac-25

C&DH Design Process

1. Allocate mission and system requirements
2. Define the computer system's operational modes and states
3. Functionally partition and allocate the computational requirements
    1. Decide what part of the system architecture will be responsible for each computational requirement. For example, consider allocating functions to space vs. ground, payload, the spacecraft bus, or other subsystems. Distinguish between hardware and software requirements.
4. Evaluate internal and external interfaces
5. Select baseline architecture
6. Form the baseline system specification

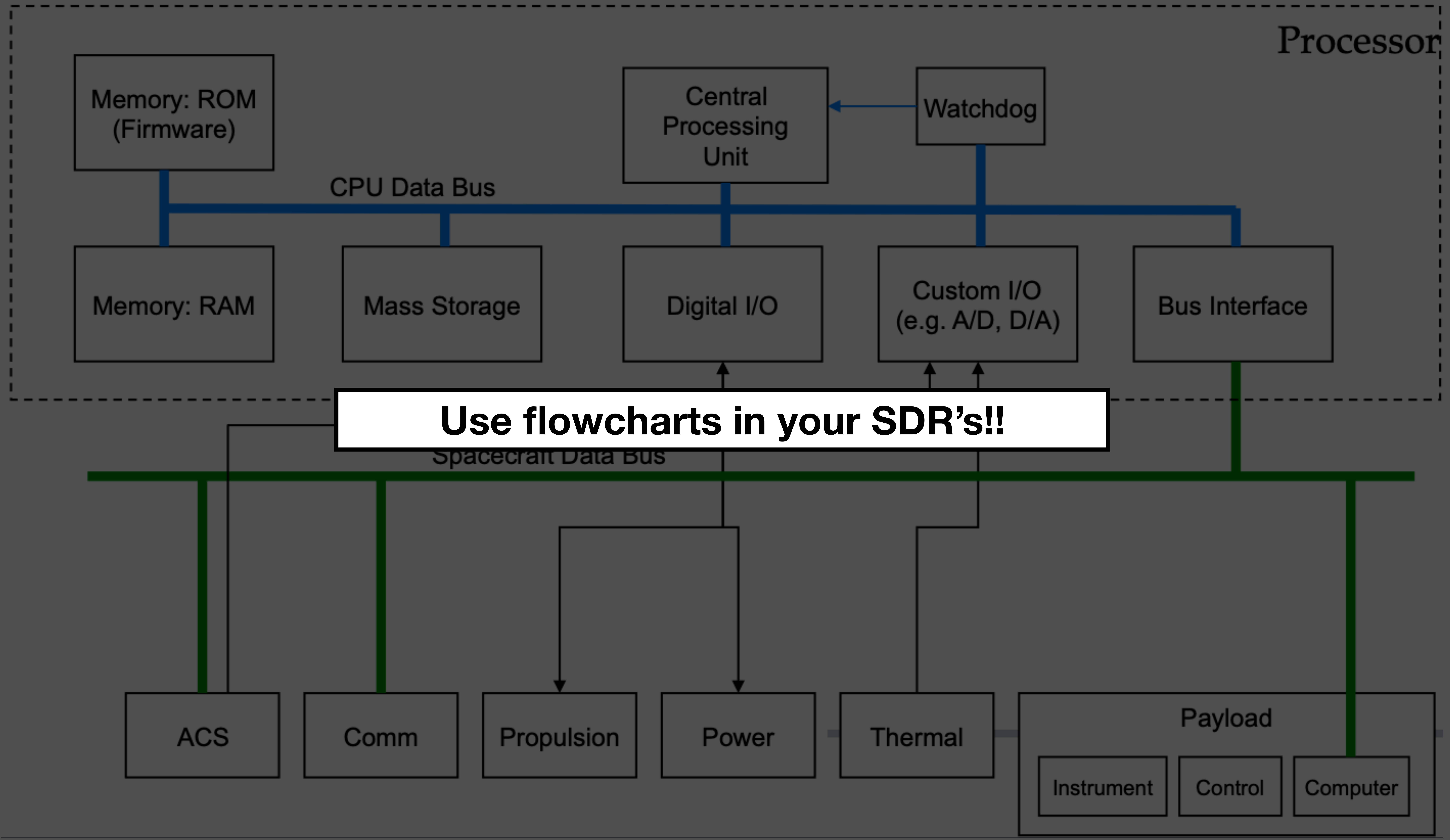
C&DH Design Process

1. Allocate mission and system requirements
2. Define the computer system's operational modes and states
3. Functionally partition and allocate the computational requirements
4. Evaluate internal and external interfaces
    1. Determine input/output requirements for the avionics subystem with respect to the other subsystems and payload.
5. Select baseline architecture
6. Form the baseline system specification



Processor
Memory: ROM (Firmware)
Central Processing Unit
Watchdog
CPU Data Bus
Memory: RAM
Mass Storage
Digital I/O
Custom I/O (e.g. A/D, D/A)
Bus Interface
Spacecraft Data Bus
ACS
Comm
Propulsion
Power
Thermal
Payload
Instrument
Control
Computer

Use flowcharts in your SDR's!!



C&DH Design Process

1. Allocate mission and system requirements
2. Define the computer system’s operational modes and states
3. Functionally partition and allocate the computational requirements
4. Evaluate internal and external interfaces
5. Select baseline architecture
    1. Will your system use centralized or distributed processing? If distributed, what type? Do you need redundancy? - *More on this a bit later*
6. Form the baseline system specification

C&DH Design Process

1. Allocate mission and system requirements
2. Define the computer system’s operational modes and states
3. Functionally partition and allocate the computational requirements
4. Evaluate internal and external interfaces
5. Select baseline architecture
6. Form the baseline system specification
    1. Create a detailed design and integration, assembly & test strategy.

Today’s topics:

- Apollo flight computer
- Avionics design process
- **Avionics technologies**
- Reliability and architecture options
- What could go wrong?

Avionics technologies

1. Memory
2. Mass data storage
3. Input/Output
4. Processors

Avionics technologies: **Memory**

Memory is just a mechanism for storing data, which is just 1's and 0’s.

Flavors:

Read-only memory (ROM): Non-volatile, so data remains regardless of power or reset. Slow write speeds, interfaces with the CPU for read-only purposes (lookup tables, program information, etc.). Usually devoted to **firmware**, software which will not change. Often reprogrammable with special instructions/voltages.

Random-access memory (RAM): This memory is volatile, which means that data is lost when power is removed and/or reset. It is also fast, up to over 1000 MHz. This is the memory that your software uses to store variables while you're doing math, temporarily store sensor readings, etc. See also the Daft Punk album.

Random-access memory (RAM): This memory is non-volatile, just like ROM, but you can both write to it and read from it. This memory will often store "software/firmware" that is expected to be updated, but otherwise does not change during normal operations. Flash memory is an example of this. From a reliability standpoint, it is a good idea to make your spacecraft remotely reprogrammable. One way to do that is to store a bootloader in ROM, and store the program in NVRAM so that it can be rewritten. The chipsats are capable of this.

Avionics technologies: **Mass storage**

Memory is just a mechanism for storing data, which is just 1's and 0’s.

Flavors:

Hard disk: Based on mechanical/magnetic rotating disks. A common available technology with the largest storage capacities, *but* carry concerns surrounding reliability, angular momentum, and vibration.

Magnetic/optical/digital tapes: An old mechanical system with highly reliable data storage, but which uses a stack structure (first in, last out). This leads to slow data access, making this sort of memory generally unuseful for operating systems/programs.

Bubble memory: Solid state and non-volatile with no moving parts. Patterns in magnetic permalloy and rotating magnetic field cause bubbles to move under a read/write head. The data persists in a magnetic field bias, but this systems are high mass, have lots of power dissipation, and generate unwanted magnetic fields.

Solid state drives with flash memory: Most common today.

Avionics technologies: **Mass storage**

Memory is just a mechanism for storing data, which is just 1's and 0’s.

Flavors:

Hard disk: Based or e technology with the largest storage capa nomentum, and vibration.

Magnetic/optical/d e data storage, but which uses a stack s making this sort of memory generally ur

Bubble memory: So in magnetic permalloy and rotating magnet The data persists in a magnetic field bias, ation, and generate unwanted magnetic

Solid state drives w

Avionics technologies: **Mass storage**

Memory is just a mechanism for storing data, which is just 1's and 0’s.

Flavors:

Hard disk: Based on mechanical/magnetic rotating disks. A common available technology with the largest storage capacities, *but* carry concerns surrounding reliability, angular momentum, and vibration.

Magnetic/optical/digital tapes: An old mechanical system with highly reliable data storage, but which uses a stack structure (first in, last out). This leads to slow data access, making this sort of memory generally unuseful for operating systems/programs.

Bubble memory: Solid state and non-volatile with no moving parts. Patterns in magnetic permalloy and rotating magnetic field cause bubbles to move under a read/write head. The data persists in a magnetic field bias, but this systems are high mass, have lots of power dissipation, and generate unwanted magnetic fields.

Solid state drives with flash memory: Most common today.

Avionics technologies: **Mass storage**

Memory is just a mechanism for storing data, which is just 1's and 0’s.

Flavors:

Hard disk: Based on me le technology with the largest storage capacitie momentum, and vibration.

Magnetic/optical/digita le data storage, but which uses a stack struc , making this sort of memory generally unuse

Bubble memory: Solid s in magnetic permalloy and rotating magnetic fie The data persists in a magnetic field bias, but t pation, and generate unwanted magnetic field

Solid state drives with

Avionics technologies: **Mass storage**

Memory is just a mechanism for storing data, which is just 1's and 0’s.

Flavors:

Hard disk: Based on mechanical/magnetic rotating disks. A common available technology with the largest storage capacities, *but* carry concerns surrounding reliability, angular momentum, and vibration.

Magnetic/optical/digital tapes: An old mechanical system with highly reliable data storage, but which uses a stack structure (first in, last out). This leads to slow data access, making this sort of memory generally unuseful for operating systems/programs.

Bubble memory: Solid state and non-volatile with no moving parts. Patterns in magnetic permalloy and rotating magnetic field cause bubbles to move under a read/write head. The data persists in a magnetic field bias, but this systems are high mass, have lots of power dissipation, and generate unwanted magnetic fields.

Solid state drives with flash memory: Most common today.

Avionics technologies: **Mass storage**

Memory is just a mechanism for storing data, which is just 1's and 0’s.

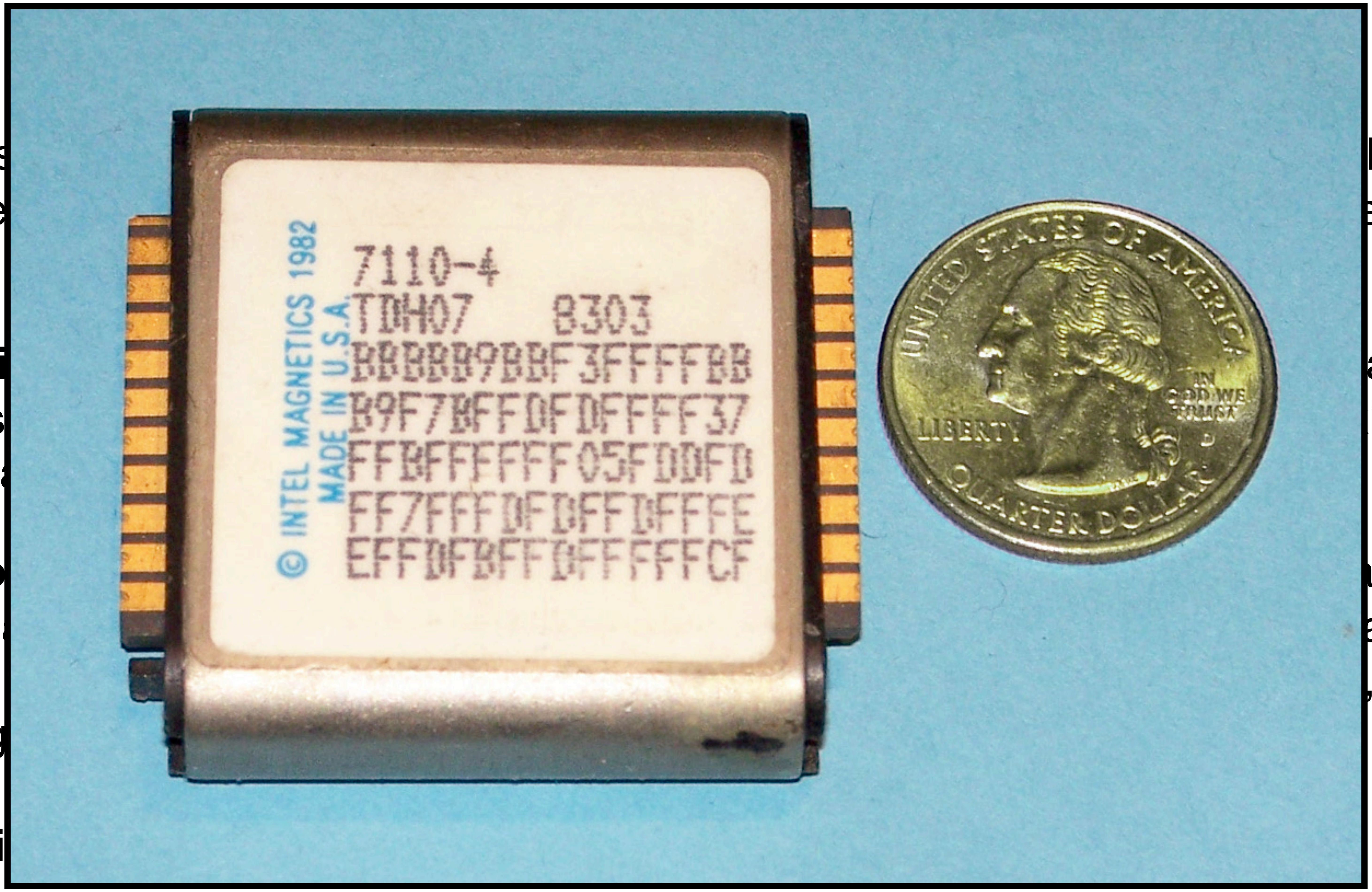
Flavors:

Hard disk: Bas hnology with the largest storage entum, and vibration.

Magnetic/opti a storage, but which uses a s ing this sort of memory gener

Bubble memo gnetic permalloy and rotating m ata persists in a magnetic field , and generate unwanted mag

Solid state dri



Avionics technologies: **Input/output**

Data bus: Reserved for high speed (>500 Mbps) data transfers among all subsystems that generate and receive data/commands.

Digital I/O: Interfaces directly with the CPU. For mapped I/O, the CPU program may directly address these ports. This enables direct control of parallel digital signals at low bandwidth.

Custom I/O: Includes things like analog to digital converters and specialized ports (UART, I2C, USB, etc.)

Avionics technologies: **Processors**

Microcontrollers (MCU): These are small, dedicated processors for performing very specific tasks (interfacing with a single sensor, for example, or actuating a valve). They run at 1-100 MHz, <16MB RAM, ROM storage (no HD/mass storage)

Digital signal processors (DSP): Specifically designed to manage embedded digital systems, and have a high processing/power radio (e.g. 1 GIPS @ 6W). These typically run 100-1000 MHz, 16-2000 MB RAM, ROM storage, and FLASH storage.

Microprocessors (μP): Powerful processors, with with a high power consumption. >1 GIPS, 50-200W, 4GB+RAM, ROM storage, HD's, CD-ROM's, etc. These have a general purpose instruction set.

Today’s topics:

- Apollo flight computer
- Avionics design process
- Avionics technologies
- **Reliability and architecture options**
- What could go wrong?

Centralized vs. **Distributed** Processing

Centralized: One processor designated as master unit, which provides all housekeeping and data handling. All commands are processed/routed through this central unit.

Distributed: Multiple processors divide the avionics tasks. There are two possible configurations: *distributed computing* and *redundant processing*.

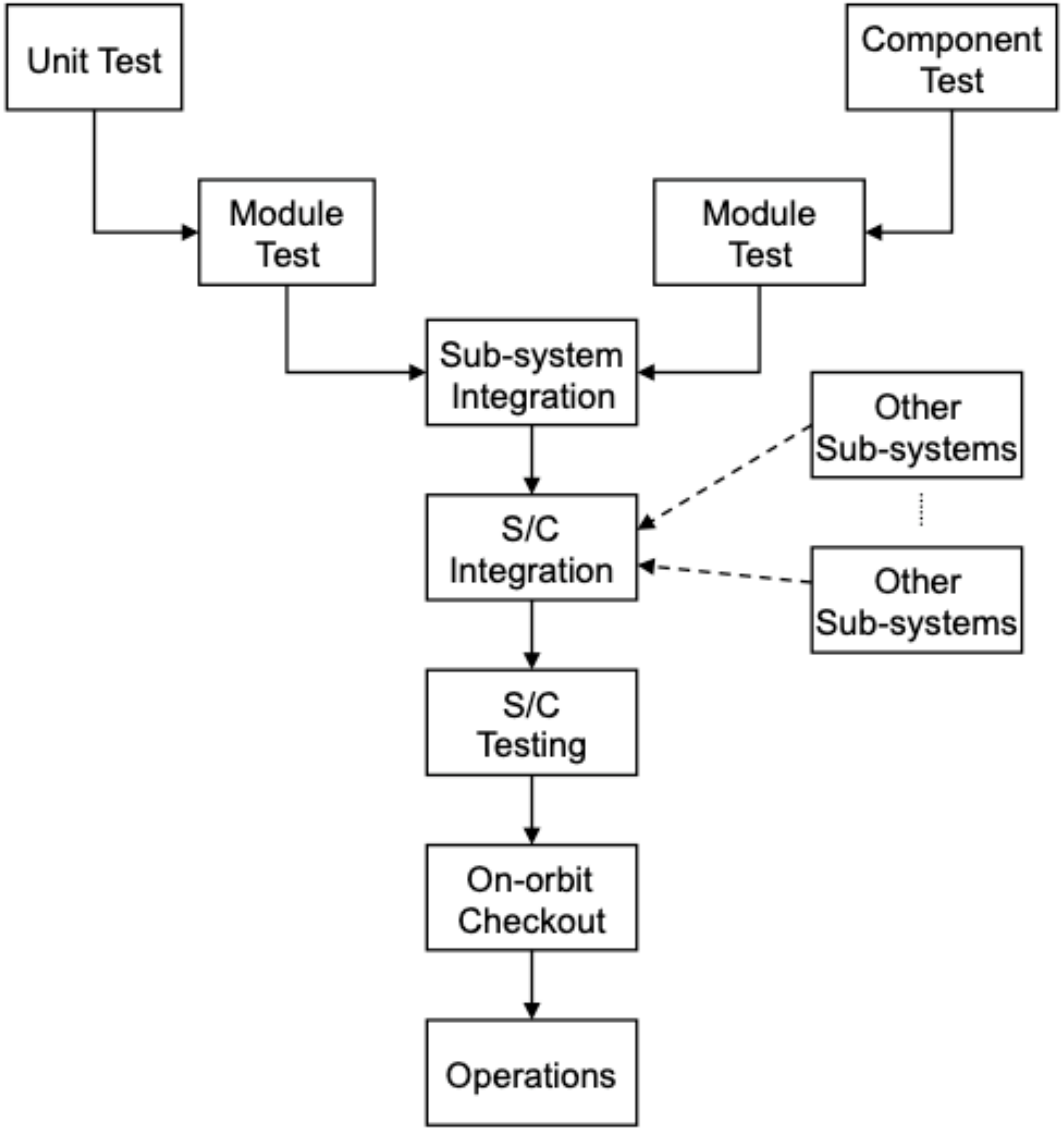
In distributed computing, executive tasks are shared by all processors, and dedicated processors are assigned to each subsystem. All of these processors communicate over the spacecraft bus. In redundant processing, multiple processors can assume the role of masters. This architecture tolerates faults well.

Radiation Hardening

- High-energy neutrons will cause structural damage to solid state materials.
- Charge on gates of metal-oxide semiconductors can change their state for single-event upsets.
- Clouds of electrical charge can slow digital logic, alter op-amp offset voltages, reduce current capability, and latch CMOS gates.

What to do?

- Shield electronics boxes
- Coat electronics with radiation-resistant materials
- Re-design electronics to a more fault-tolerant architecture.
- *Costs a lot more money.*



Testing

Always always always test bottom-up**.**

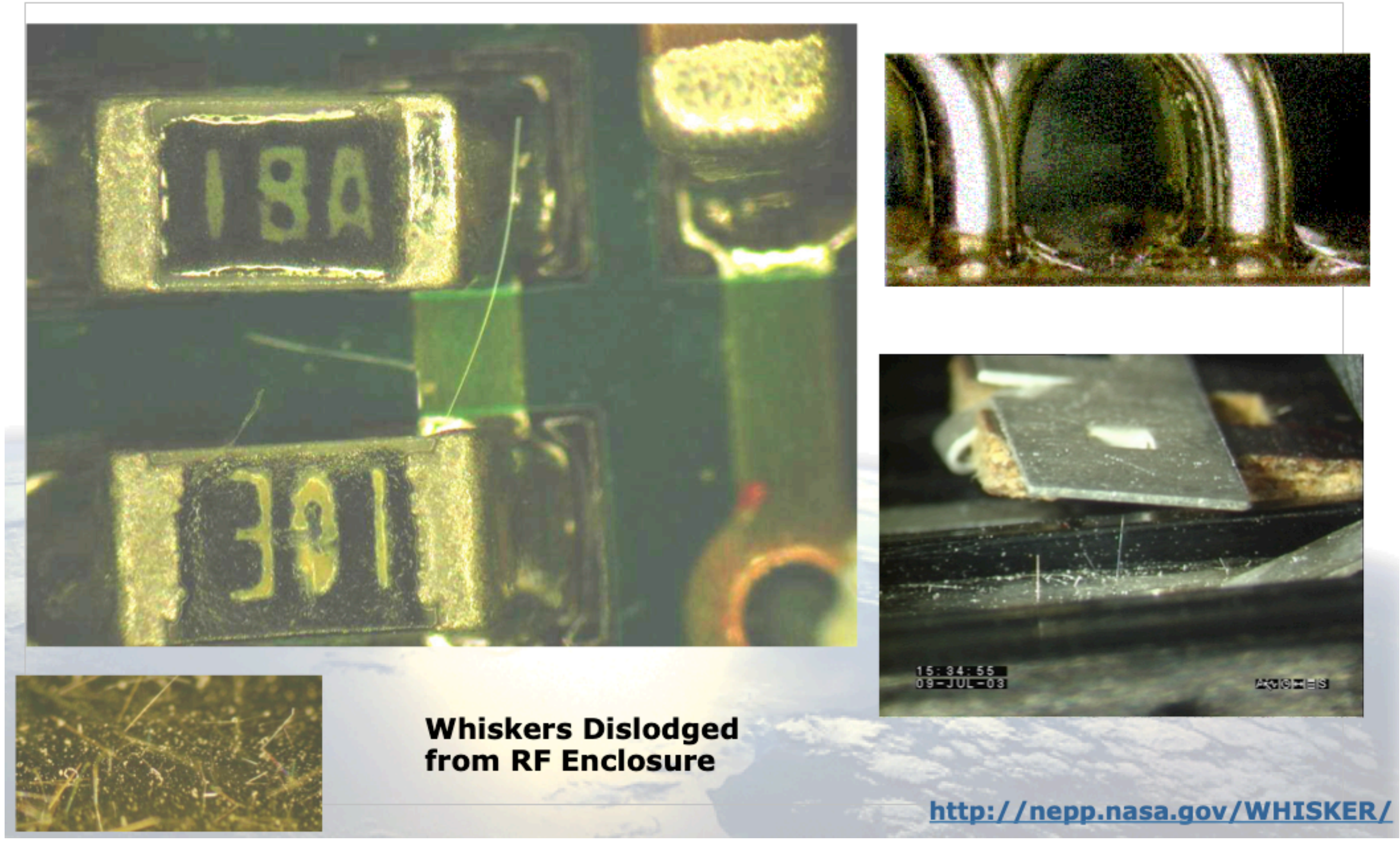
Today's topics:

- Apollo flight computer
- Avionics design process
- Avionics technologies
- Reliability and architecture options
- **What could go wrong?**

Tin Whiskers

Pure zinc and tin can form “whiskers,” dendritic crystal growths which can cause short circuits.

Solution: don't use pure zinc/tin solder

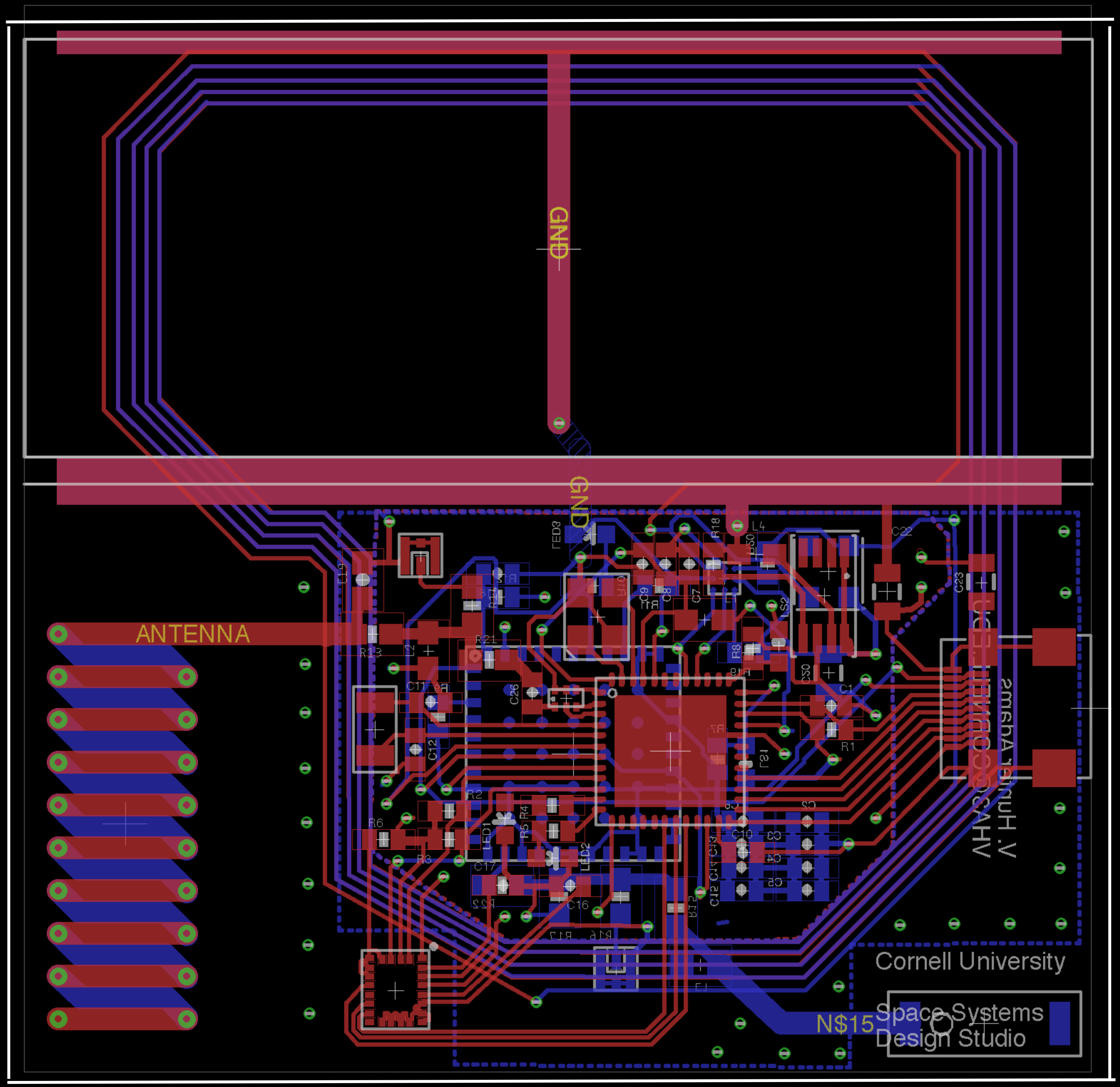


Galaxy IV (Launched 6/24/93)

- **HS-601 GEO communications spacecraft that carried about 1/3 of the nation's pager traffic, among other things.**
- Attitude control failed 5/19/98 when the satellite's primary control processor failed. The backup control processor had suffered a previously undetected anomaly.
- Declared a loss on May 20, 1998.

- A hole had developed in the conformal wax coating over the tin solder, allowing tin whiskers to develop. The satellite manufacturer, now Boeing, has replaced pure tin with nickel to alleviate the problem in newer designs, adding 45 to 90 kg per spacecraft.

monarch avionics



software

bare metal programming

vs.

real-time operating system

bare metal programming

1. initialization (runs once)

2. state machine
   (runs continuously)

3. interrupt service routine
   (runs once per interrupt,
   often changes state)

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(
                  interruptPin), blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

bare metal programming

small, fast, and easy to understand for simple applications

but!

developer must handle power, memory, and interrupt table management

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(
                  interruptPin), blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

not scalable, **not** modular, but have their place

real-time operating system

1. Initialization (runs once)

2. Task creation (runs once)

3. Call to scheduler
   (where the magic happens)

```c
int main(void)
{
  /* Initialize TI drivers */
    Board_initGeneral();
    PIN_init(pinTable);

    /* Setup peripherals and semaphores */
    wdtSetup();
    clockSetup();
    semaphoreSetup();
    pinSetup();


    /* Construct tasks */
    createMagTask();
    createGyroTask();
    createAccelTask();
    createGPSTask();
    createADCTask();
    createRFRXTasks();
    createRFTXTasks();
    createPWMTask();


    /* Start kernel. */
    BIOS_start();

    return (0);
}
```

real-time operating system

rule of thumb

if your application needs to do more than a few simple actions, or if you have any intention of scaling up your application, or of having multiple people develop simultaneously…

use an RTOS!

Brief look at this

```
int main(void)
{
 /* Initialize TI drivers */
    Board_initGeneral();
    PIN_init(pinTable);

    /* Setup peripherals and semaphores */
    wdtSetup();
    clockSetup();
    semaphoreSetup();
    pinSetup();


    /* Construct tasks */
    createMagTask();
    createGyroTask();
    createAccelTask();
    createGPSTask();
    createADCTask();
    createRFRXTasks();
    createRFTXTasks();
    createPWMTask();


    /* Start kernel. */
    BIOS_start();

    return (0);
}
```

real-time operating system

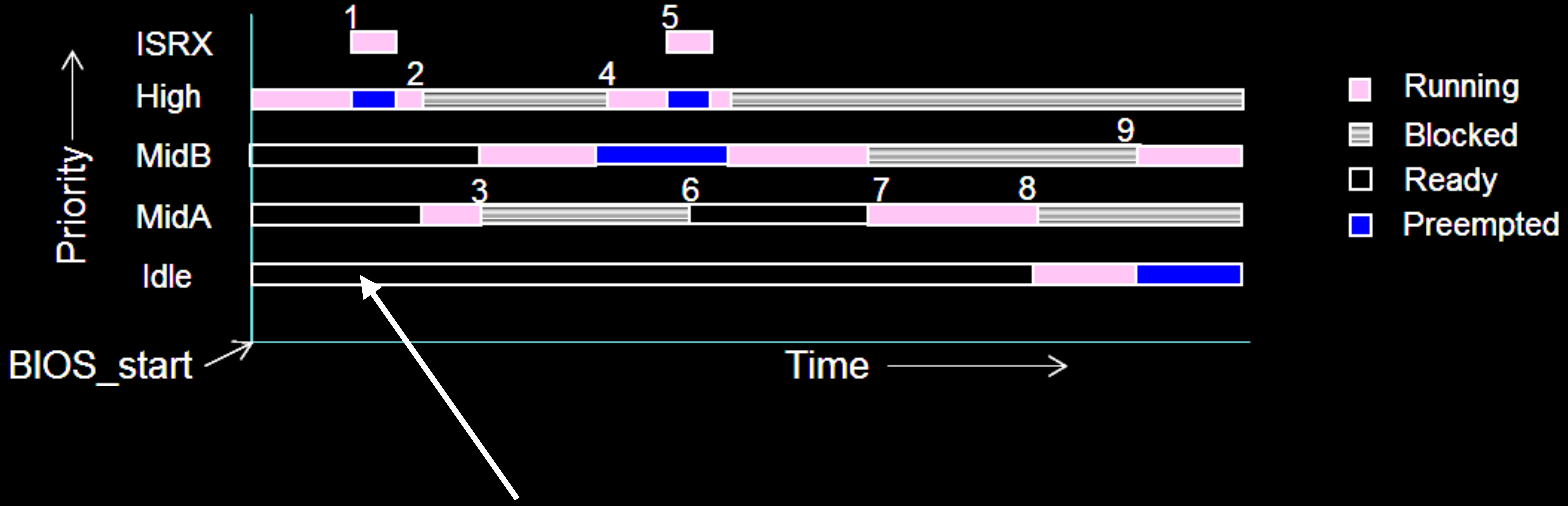
Interrupt Service Routine: Thread initiated by hardware interrupt. Asserts and runs to completion. *Rule of thumb*: get in and get out as quickly as possible.

Tasks: Thread that can block while waiting for an event to occur. Traditionally long-living threads (as opposed to ISRs which run to completion). Each task has it's own stack which allows it to be long living.

Preemptive Scheduler: A scheduler in which a running thread continues until:

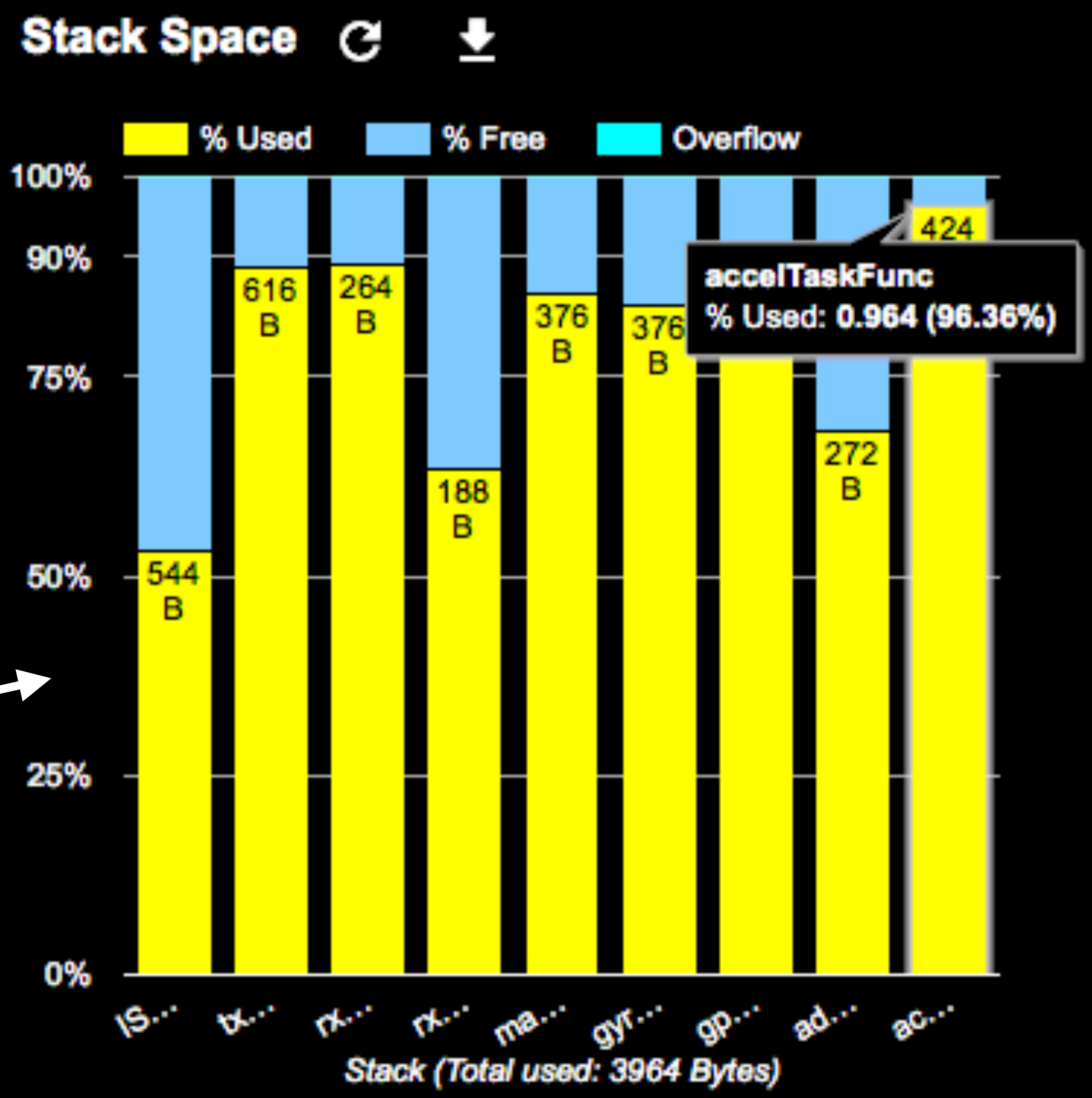
- It finishes (e.g. an ISR completes)
- A higher priority thread becomes ready ("preempts" lower priority thread)
- Thread yields processor while waiting for a resource.

real-time operating system



automatically places MCU in a low-power mode

real-time operating system



Automatically manages memory

real-time operating system

Semaphore: Method for thread communication that allows for resource management. Can be thought of like a baton or a speaking stick, whoever controls the semaphore controls the CPU.

Thread-safe: A piece of code is thread-safe if it manipulates shared data structures in a manner that guarantees correct access (reading/writing) by multiple threads at the same time.

```
Void myISR() {
   // get data from
   // peripheral

   sem_post();

   // finish
}
```

```
Void myTask() {
  …
  while(cond) {
     sem_wait();
     // Process data
  }
  …
}
```

real-time operating system

```
void pinCallback(PIN_Handle handle, PIN_Id
                 pinId) {
    uint32_t currVal = 0;
 switch (pinId) {
   case CC1310_LAUNCHXL_DIO12:
      Semaphore_post(gyroSemaphoreHandle);
      break;

   case IOID_14:
      Semaphore_post(magSemaphoreHandle);
      break;

   case IOID_13:
      Semaphore_post(accelSemaphoreHandle);
      break;

   case IOID_1:
      currVal =  PIN_getOutputValue(
              CC1310_LAUNCHXL_PIN_RLED);
      PIN_setOutputValue(pinHandle,
                      CC1310_LAUNCHXL_PIN_RLED,
                      !currVal);
      break;


   default:
      break;
 }
}
```

mutex for i2c access

```
Void gyroTaskFunc(UArg arg0, UArg arg1)
{
    while (1) {
        Semaphore_pend(gyroSemaphoreHandle,
                    BIOS_WAIT_FOREVER);
        Semaphore_pend(batonSemaphoreHandle,
                    BIOS_WAIT_FOREVER);
        if(goodToGo){
          readGyro();
        }
        Semaphore_post(batonSemaphoreHandle);
    }
}

Void accelTaskFunc(UArg arg0, UArg arg1)
{
    while (1) {
        Semaphore_pend(accelSemaphoreHandle,
                    BIOS_WAIT_FOREVER);
        Semaphore_pend(batonSemaphoreHandle,
                    BIOS_WAIT_FOREVER);
        if(goodToGo){
          readAccel();
        }
        Semaphore_post(batonSemaphoreHandle);
    }
}
```

monarch software architecture

(Videos on my YouTube channel, code on GitHub)

Hardware Interrupts	High Priority Tasks	Low Priority Tasks	Outside RTOS
• Gyro data ready	• Retrieve gyro data	• Read ADC data	
• Accel data ready	• Retrieve accel data	• Attitude determination	
• Mag data ready	• Retrieve mag data	• PWM torque coils	• Reset CPU (watchdog)
• TX timer timeout	• Transmit beacon	• Map neighbors	• GPS/Radio RX timer
• Radio data in buffer	• Retrieve radio data	• Retrieve GPS data	
• UART data in buffer	• Reset watchdog		

monarch software architecture

Fundamental Limitation

128 kB of programmable flash memory

What's Still Missing

Routing over the network

Global knowledge: dynamic programming problem
Local knowledge: optimal stopping problem (?)