Vodafone CrowdCell Course:

Radio and Programming Considerations

Lime Microsystems| FPRF company
Guildford, Surrey, United Kingdom

Introduction

Part 1 RF considerations

- What is the range of my LimeNet Mini or Crowd Cell?
- Living with Bad Neighbours

Part 2 Programming considerations

- Properties of open source programming computer languages
- Vector Maths

RF Considerations

1

Part 1. RF Considerations

It is easy to think a SDR as just a USB dongle.

For best performance, practical radios require additional components

- Amplifiers and Antennas for best range.
- Filters to prevent interference problems.

Lets look in a bit more detail at two key considerations

- SDR Range.
- The interference problem.

What is the range of a SDR

A very simple question…

- **Without a simple answer…**

Depends on many things…

- **Distance between Tx and Rx**
- **Height and gain of Tx and Rx antennas**
- **Frequency and Bandwidth of signal**
- **Level of Forward Error Correction coding**
- **RF matching of Rx and its NF**
- **Does the signal have to pass through trees, walls and buildings?**
- **Planes, trains and automobiles.**
- **Landscape, weather and even the solar system!**

Classic approach is the Link Budget

- **Depends on…**

Tx Output power

Path Loss

- **Distance and frequency**

Antenna gains, Duplexing filters etc.

Rx Sensitivity

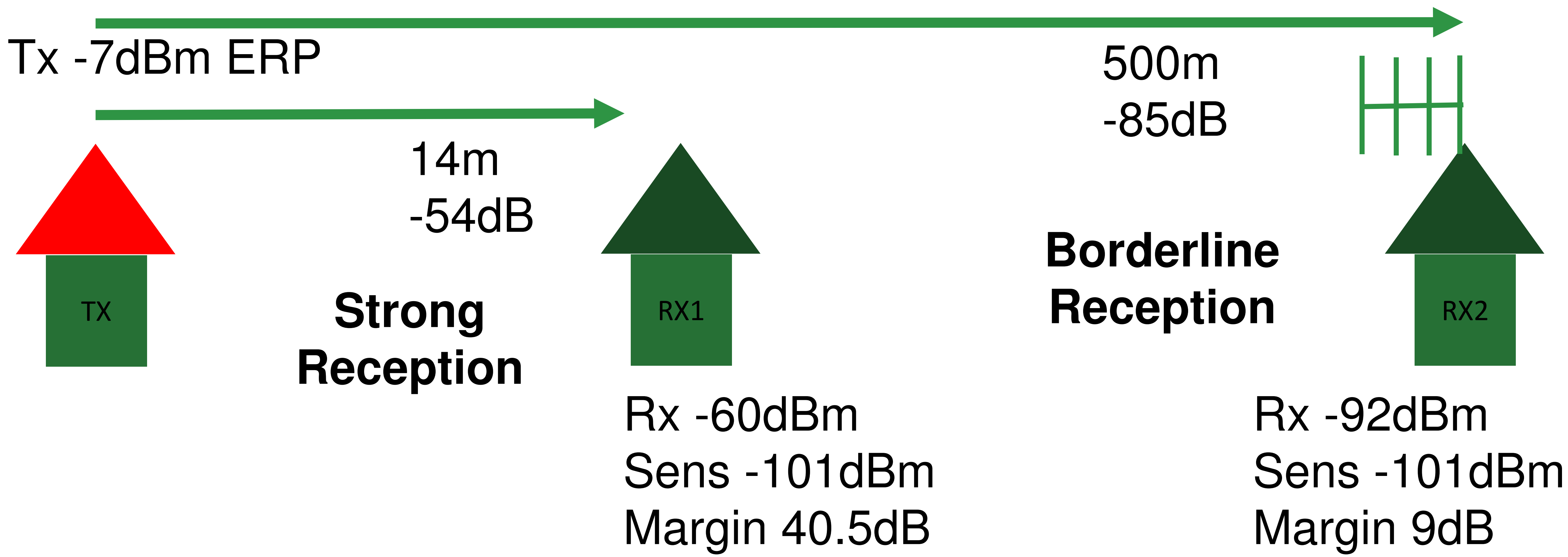
- **Bandwidth, Forward Error correction and Receiver noise figure**

Fading behaviour

- **Depends on height, speed etc. Need some margin – usually 40dB for mobile e.g. Hata model.**

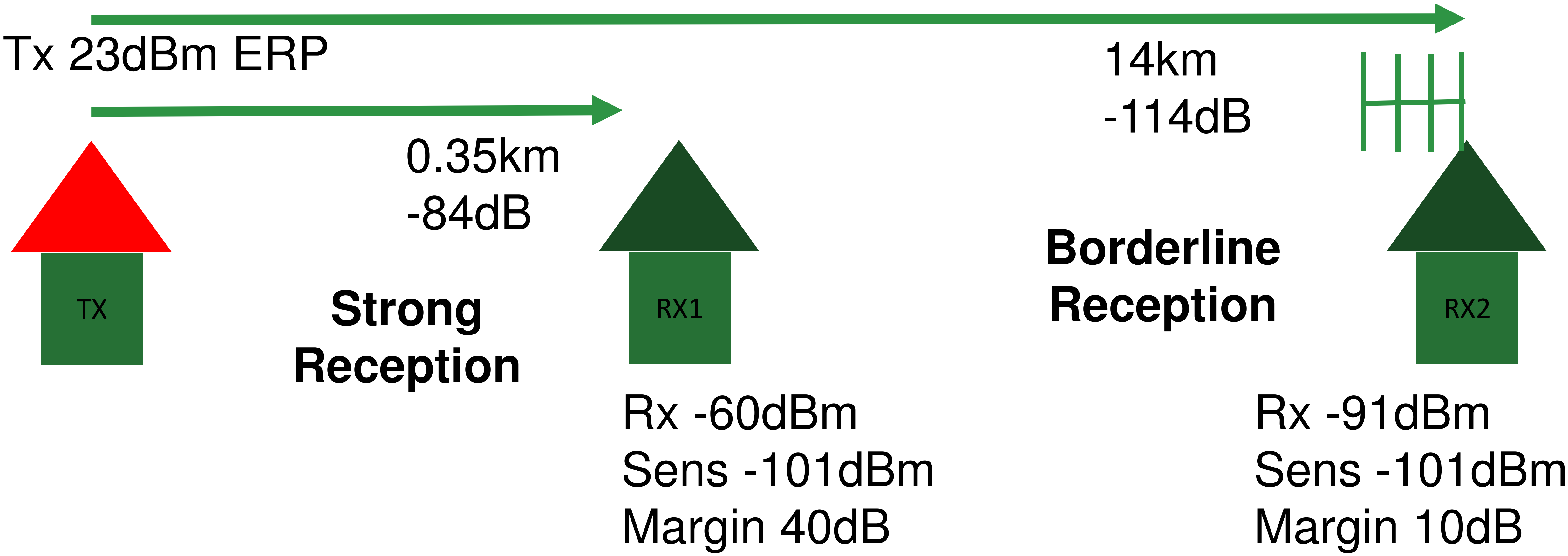
LimeNet Mini 5MHz LTE Radio Link Budget

Range of a Radio is usually a probability not a certainty!



Crowd Cell 5MHz LTE Radio Link Budget

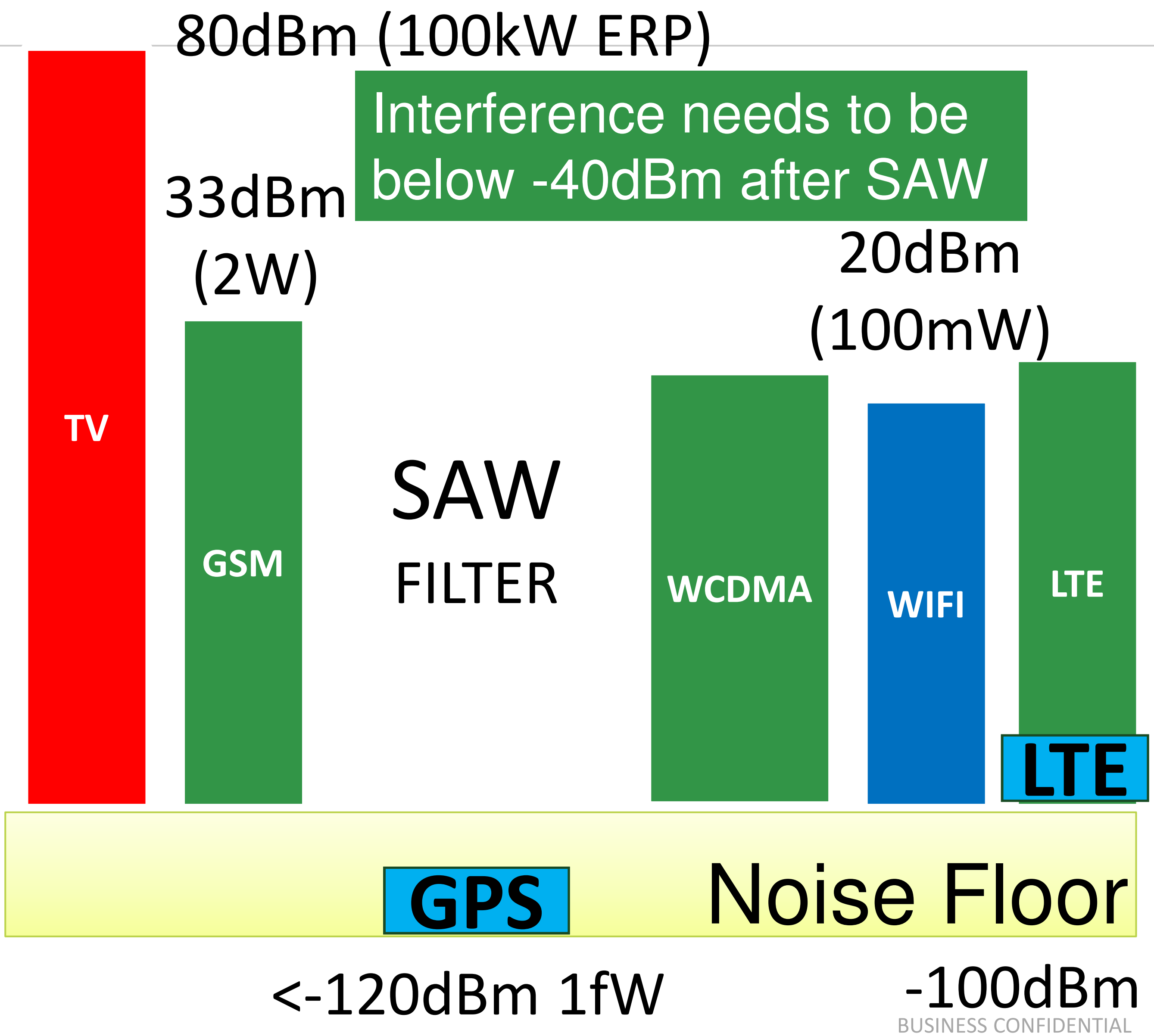
Range of a Radio is usually a probability not a certainty!



Typical Ranges for CrowdCell and LimeNet Mini

	LimeNetMini	LimeNetMini	LimeNetMini	CrowdCell	CrowdCell	CrowdCell	CrowdCell	
LimeSDR	**LTE (4G) 5M FDD**	**LTE (4G) 5M FDD**	**LTE (4G) 10M FDD**	**LTE (4G) 5M FDD**	**LTE (4G) 5M FDD**	**LTE (4G) 10M FDD**	**LTE (4G) 10M TDD**	
	OFDM-QPSK	OFDM-QPSK	OFDM-QPSK	OFDM-QPSK	OFDM-QPSK	OFDM-QPSK	OFDM-QPSK	
LO	870.00	870.00	870.00	870.00	870.00	870.00	2450.00	MHz
RF BW	4.50	4.50	9.00	4.50	4.50	9.00	9.00	MHz
Tx Level	**-7.00**	**-7.00**	**-7.00**	**23.00**	**23.00**	**23.00**	**23.00**	**dBm**
Tx Filter loss	1.50	1.50	1.50	1.50	1.50	1.50	0.50	dB
TxAe	1.00	1.00	1.00	1.00	1.00	1.00	1.00	dBi
RxAe	1.00	1.00	1.00	1.00	1.00	1.00	1.00	dBi
Dist	**0.500**	**0.014**	**0.010**	**14.000**	**0.450**	**0.300**	**0.150**	**km**
Loss	85.21	54.15	51.23	114.15	84.30	80.77	83.75	dB
Other Loss	0.00	0.00	0.00	0.00	0.00	0.00	0.00	dB
Rx Level	**-91.71**	**-60.65**	**-57.73**	**-90.65**	**-60.80**	**-57.27**	**-59.25**	**dBm**
Thermal	-107.30	-107.30	-104.29	-107.30	-107.30	-104.29	-104.29	dBm
Eb/No*	-0.50	-0.50	-0.50	-0.50	-0.50	-0.50	-0.50	dB
Spread Factor	1	1	1	1	1	1	1	
CodeGain	0.00	0.00	0.00	0.00	0.00	0.00	0.00	dB
RF Switch	0.50	0.50	0.50	0.50	0.50	0.50	0.50	dB
RX Filter Loss	2.50	2.50	2.50	2.50	2.50	2.50	0.50	dB
Rx NF	4.00	4.00	4.00	4.00	4.00	4.00	4.00	dB
Sensitivity	**-100.80**	**-100.80**	**-97.79**	**-100.80**	**-100.80**	**-97.79**	**-99.79**	**dBm**
Margin	**9.09**	**40.14**	**40.06**	**10.14**	**40.00**	**40.51**	**40.54**	**dB**

Interference – Living with Bad Neighbours!



Broadcast, Mobile and WiFi leads to neighbour challenges

Receiver must be able to work with very low signals in the presence of strong interfering signals.

Requires low NF, low far out phase noise (-160dBc), high P1dB and good IIP2 and IIP3 and a good ADC.

With FDD, your own TX is your worst interferer! Need SAW filters radio.

Lime microsystems

2

Programming Considerations

Part 2. Programming Considerations:

Software defined radio needs efficient software as well as low cost hardware.

Low cost processors give high performance vector maths capability needed for SDR.

- I3-8100 Quad Core, SSE4.2, AVX2, FMA3 $120
- ARM A8 of Raspberry Pi includes NEON vector maths. $30

Rise of the open source software movement

- Linux and Android systems support open source software.
- Wide range of mature and emerging open source programming languages
  - Most languages supports files, pipes, complex numbers, TCP sockets etc.
- Wide range of speed optimized libraries such as VOLK, BLAS, FFTW etc.
- Visualisation tools such as GNUPLOT pipes.

Moving away from custom ASICs, expensive FPGA to low cost SDR.

So which programming languages are good for my SDR?...

Which Programming Language for my SDR?

Tradeoffs

- Development Time, Execution Speed & Interconnectivity with other languages

C/C++, Fortran95

- Software for high speed real time links.
- C/C++ has excellent static analysis tools
- Debug time can be slow

Octave/Matlab etc.

- Easy to use highly interactive languages for education and for R&D
- Efficient algorithm development
- Easy visualisation tools GNUPLOT

GNU Radio, Pothos

- Integrated environments for 'real time' software defined radio systems.
- Precompiled modules BLAS, VOLK, FFTW etc.

Python/Ruby

- Interactive languages
- Compact, good for GUI and Networks
- Not usually good for high speed.

Perl/PHP/Java/JavaScript etc.

- Scripting and remote database links.

ADA

- Safety Critical Systems.
- Extended Error checking

Languages from SDR Perspective...

Lime microsystems

GNU	TCP + UDP Sockets	Linux Sys Calls	ARGV	Text Files	Binary Files	Pipe Output	Complex Numbers	Links to C	TK	Irregular Lists	Compiled?
ADA	Yes?	via C?	YES	YES	YES	YES	?	?	?	?	YES
C/C++	YES	YES	YES	YES	YES	YES	YES	YES	YES	USER DEF	YES
Fortran95	Yes?	Send only	YES	YES	YES	YES	Vector/Matrix	YES	Ex C?	NO	YES
Com LISP	Yes?	YES	NonStd	YES	YES	YES	YES	??	Ex Lib	YES	Optional
Java	YES	YES	YES	YES	YES	YES	USER DEF	?	?	USER DEF?	YES
Julia	YES	YES	YES	YES	YES	YES	Vector/Matrix/FFT	Shared Lib	?	?	Interpreted?
Octave	YES	YES	YES	YES	YES	YES	Vector/Matrix/FFT	YES	?	NO	Optional?
Perl	YES	YES	YES	YES	YES	YES	YES	Shared Lib	YES	NO	Interpreted?
Prolog	Yes?	?	?	?	?	interactive	USER DEF?	?	?	?	Interactive
Python	YES	YES	YES	YES	YES	YES	Vector/Matrix/FFT	YES	YES	YES	Optional
Ruby	YES	YES	YES	YES	YES	YES	YES	Shared Lib	YES	NO	Inteprted?

Most languages do sockets, system calls, command lines, files and pipes.
How *PAINFUL* is it to link between languages? Language should define basic constants.
Artificial intelligence needs irregular and flexible data structures for real life.
Language independent GUI can help speed development.
User defined structures often have additional burdens that slow down programming.

Languages from SDR perspective…

		(-Ofast)	(-Ofast)	(Numpy)	(Repository)	(Repository)
Time in ms	**Target Time**	**C 99**	**FORTRAN 95**	**PYTHON 2.7**	**OCTAVE 4.0.3**	**JULIA 0.4.5**
BPSK Hadamard 256 build (W-CDMA)	initial	0.517	1	6.1	15	763
Binary PRS 4096 (WiFi 802.11a)	0.1	0.038	2	153	4465	80.4
Complex Number Spread 300*256 (W-CDMA CPICH)	0.1	2.615	3	2.8	13	4204
Complex Number Despread 300*256 (W-CDMA CPICH)	0.1	0.579	1	2.1	18	0.605
RRC FIR Complex Number 76800pts x4 OSR +/-3symbols	1	0.059	125	1005		9231
Complex Number FFT 4096 points	0.2	0.806	1	2.176	24	1496
Software tested on a 1.44GHz Z83 Atom with Ubuntu 16.04 Low Latency		Fortran was originally developed by IBM in the 1950s				
Vectorisation used where possible for speed		C was originally developed by Bell Labs in the 1970s.				
Functions inlined for speed		Octave is a GNU clone of the Matlab language developed in 1970s				
GNU compilers gfortran5, g++5		Python was originally developed in the Netherlands in the 1990s				
Timing not usually 7repeatable, best of 3 runs		Julia began development in 2009				
Fortran CPU_TIME only measures to 1ms accuracy						
Libfftw3.3 installed on ubuntu						

Languages – Simple FSK Example

OCTAVE

- 26 Lines, including graph plot
- About 20mins to code and debug

FORTRAN90

- 86 Lines
- about 3 hours to code and debug

C99 (Complex numbers)

- 110 Lines,
- about 3 hours to code and debug

(Shorter line counts possible at expense of readability)

High level numerical languages

- Vectorised maths is compact to code.
- 'roots' in Fortran,
- So map more easily to FORTRAN than C
- Interactive, speeds up prototyping

FORTRAN

- Fast, Compiler optimised for large data objects.
- Compiler error messages often obtuse.

C compilers

- Very fast, heavily optimised for speed.
- Work very well with small data objects.
- Compiler much more 'static analysis' aware.
- Unsafe programming structures easily used

Vector Maths 1

SIMD4.2 128 bit. (Most Atom,Pentium etc)

- Parallel array 16x 8bit char, 8x 16bit short or 4x 32bit int.
- Speeds up integer vector maths such as CRCs and PRSs

AVX2/FMA3 256 bit. (4th gen i3/i5/i7, AMD Bulldozer/PileDriver)

- AVX Parallel array 8x 32bit Floats, or 4x 64 bit doubles
- Speeds up vector operations such as FFTs
- AVX2 makes SIMD equivalent of 256 bit
- FMA3 a=a.b+c type maths (e.g. radix pairs used in FFTs)

AVX512 512 bit. (Some 6th gen i7 and many i9 – not widely available)

NEON 64/128 bit (ARM A8)

- Has been issues with compiler integration.

Vector Maths 2

The Compiler, it is both your friend and your enemy!

- Obscure special flags to enable Vector maths.
- Compiler will optimize your code, but only if you write it in the right way!
- ELSE: write non-portable assembly-level code… (Strongly not recommended)

Typical compiler settings

- March=native
- Ftree-vectorise

Languages

- Use C/C++ or modern Fortran and link to higher level languages
- Octave/Matlab use Fortran notation to denote vector maths operations
- But speed up is not as good as raw C/C++ or Fortran due to complicated data inter-conversions.

Vector Maths 3 – Gold Code Example 1

PRS are based on feedback shift registers.

- Gold codes use parallel PRSs to generate longer sequences.
- E.g. LTE scramble code
- Cannot precompute as changes with message length and type.
- Avoid BPSK forms

Treat as an array…

- 4x 32bit integers – minimise register loads
- Parallel shifts on accumulators
- Parallel XOR masks to make feedback
- Parallel 'bit counts' on feedback registers
- XOR Accumulator 1 and 2 with output masks to give bitstream

SIMD Register	Literal Masks
Accumulator 1	Feedback Mask 1
Accumulator 2	Feedback Mask 2
Feedback 1	Output Mask 1
Feedback 2	Output Mask 2

Vector Maths 4 – Gold Code Example 2

```
Uint_32t myReg[4]={init1,init2,0,0};
For( cc= … )
{
 myReg[2]=myReg[0]^MASK1;
 myReg[3]=myReg[1]^MASK2;
 myReg[2]=BitCount(myReg[2]);
 myReg[3]=BitCount(myReg[3]);
 myReg[0]>>=1;
 myReg[1]>>=1;
 myReg[0]+=(mult*myReg[2]);
 myReg[1]+=(mult*myReg[3]);
 out[cc]=(myReg[0]&1)^(myReg[0]&1);
}
```

// Help compiler by using vector instead of multiple separate variables which have to be loaded separately

// Help compiler by grouping parallel commands

// Use built in single instruction bit count, DO NOT write your own using 'for loop'.

// Avoid 'hidden' intermediate results by not using a=a>>1 form.

// multiplication by 2^31 faster than <<31

// avoid using wrong kind of number in literals e.g. 1.0f, as this introduces hidden 'conversion' operators.

Summary

Although it is not really useful to specify a maximum range to a SDR, link budgets with 40dB margins generally give a good indication of useful range.

SDRs generally require internal or external filters to give best performance.

A diverse range of programming languages exist, and many can play a roll in SDR in different parts of the system.

C/C++ and Fortran are particularly suitable for high speed tasks.

Taking advantage of vector maths capabilities of modern low cost processors can significantly enhance speed of low level tasks.